

Transducer-based Rewriting Games for Active XML

Martin Schuster

TU Dortmund University

Friday 10th June, 2016

Abstract

Context-free games are two-player rewriting games that are played on nested strings representing XML documents with embedded function symbols. These games were introduced to model rewriting processes for intensional documents in the *Active XML* framework, where input documents are to be rewritten into a given target schema by calls to external services.

This paper studies the setting where dependencies between inputs and outputs of service calls are modelled by transducers, which has not been examined previously. It defines transducer models operating on nested words and studies their properties, as well as the computational complexity of the winning problem for transducer-based context-free games in several scenarios. While the complexity of this problem is quite high in most settings (ranging from NP-complete to undecidable), some tractable restrictions are also identified.

1 Introduction

Scientific context *Context-free games* on strings are two-player games extending context-free grammars, with the first player (called JULIET) choosing the non-terminal to be replaced and the second player (called ROMEO) choosing a replacement for that non-terminal. The winning condition for JULIET is reaching, at some point during the game, some string in a given target language over the combined alphabet of non-terminals and terminals.

These games were first introduced in [12] to model the rewriting process of *Active XML (AXML)* [1] documents. The intention of AXML is modelling *intensional* documents, i.e. documents that do not store all required information explicitly but instead contain references to external services, from which current information may be materialised on demand, as illustrated in the example below. To this end, AXML extends standard XML with *function nodes* referring to external web services that may be called to insert data into the AXML document when the document is requested. Context-free games abstract from AXML to model the uncertainty inherent in using external data.

A standard example (cf. [11, 12]) of an application for AXML is depicted in Figure 1. In this example, we consider (part of) an AXML document retained by a local online news site providing information about current weather and events. Initially, the server-side document looks like the one in Figure 1a. The

nodes labelled `@weather_svc` and `@events_svc` are function nodes referring to external weather and event services.

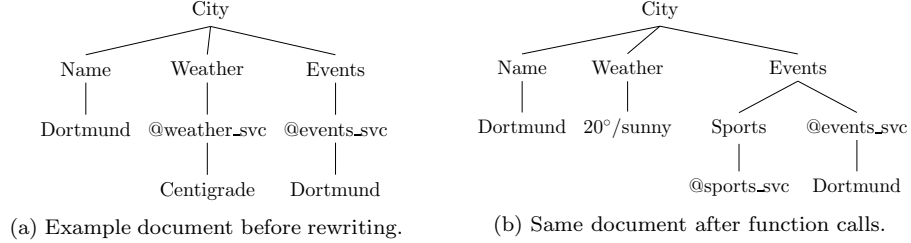


Figure 1: Example of Active XML rewriting. After calls to function nodes `@weather_svc` and `@events_svc` in Fig. 1a, external data is materialised to yield the document in Fig. 1b.

Figure 1b shows the document from Figure 1a after both function nodes have been called, replacing them by the external services' results. As exemplified by the function node labelled `@weather_svc`, call results replace the entire subtree rooted at the called function node, with that subtree being passed to the external service as a parameter. Concretely, the `@weather_svc` node's child tells the weather service that temperatures returned should be in centigrade. As the call result of the `@event_svc` node shows, returns of external services may contain further function nodes, even copies of the called function node.

The *safe rewriting problem* [11] of determining whether a given AXML document can always be rewritten into a target schema was abstracted in [12] into the problem of determining whether JULIET has a winning strategy in a given context-free game on strings, with JULIET representing a rewriting algorithm and ROMEO representing the uncertainty inherent in function calls. This research assumed DTDs as schema formalisms. Allowing more expressive schema languages such as XML Schema [14] then led to research into context-free games on *nested strings* (i.e. XML-like linearisations of trees) [3]. Even though none of these previous works modelled dependencies between parameters and outputs of function calls, they already showed that the winning problem for JULIET can be undecidable or of a very high complexity, unless strategies for JULIET and allowed schema languages are seriously restricted.

The impact of service call parameters has so far only been studied in a limited fashion. In [14] (and in [11], for AXML rewriting with DTDs), external services were modelled by *input* (or *validation*) and *output* (or *replacement*) schemas, the semantics being that a function node could only be called if its parameter subtree was valid with regard to its corresponding input schema, which would then yield a return conforming to its output schema. This is a purely syntactic handling of input parameters which models a rather simple relationship between input and output of function calls; for instance, an `@event_svc` call reproducing its input parameters in its output as shown in Figure 1 cannot be enforced in this model.

While dependencies between parameters and outputs of service calls have always been implicit in the AXML model, they have not been studied in detail so far. Therefore, we extend the context-free games on nested words from [14] by (generally non-deterministic) transformations relating function parameters to

possible outputs. We define *nested word transducers* (NWT) as a comparatively simple finite representation for transformations on nested words that naturally extends the nested word automata used in [14]. We then study the complexity of the winning problem for JULIET in various restrictions of context-free games with transducer-based replacement. As auxiliary results of potentially independent interest, we also examine closure properties and basic algorithmic problems of NWT.

Contributions In light of prior undecidability and complexity results, the main objective of this paper is finding suitable restrictions to transducer-based context-free games that render the winning problem for JULIET decidable with as low complexity as possible. The two basic types of restrictions we examine are strategy restrictions (i.e. restrictions to JULIET’s capabilities of calling function symbols) and restrictions to the type of NWT used for rewriting. To avoid undecidability, we only allow left-to-right strategies, i.e. once a function node has been called, no function calls to nodes preceding it (in post-order) are possible (cf. [12]).

The most important class of strategy restrictions considered here are restrictions to games with limited *replay*. In general context-free games, after a function call, JULIET continues her rewriting on that function call’s result; we call this the *unbounded replay* case, as JULIET may continue rewriting function call results for as long as new function nodes are returned. In the *replay-free* case, JULIET is instead forbidden to call any function nodes inside results of function calls. As an intermediate case between unbounded replay and replay-free games, we also consider *bounded replay* games, where JULIET may only call functions returned by function calls up to a fixed maximum recursion depth. For instance, in a replay-free game, JULIET could call neither of the function nodes labelled @sports_svc and @events_svc in the situation of Figure 1b; in a bounded replay game of depth 2, on the other hand, she could call these nodes, but not any function nodes returned by those secondary calls.

The second type of restrictions comes from limiting expressiveness of the transducer used in games. Generally, transducers are allowed to be *non-functional*, i.e. any input string, may have several transducts (to model the fact that function call results are dependent upon, but not uniquely determined by, input parameters). The main types of transducers examined here are the following:

- *Nested word transducers (NWT)* allow for transforming input strings into output strings that are arbitrarily long, regardless of the input string’s size.
- *Nested word transducers without ϵ -transitions (ϵ -free NWT)* may only increase the size of an input string by no more than a linear factor.
- *Relabelling transducers* may only change labels of input strings, not their structure.
- As a special case, *functional relabelling transducers* are relabelling transducers whose output string is uniquely determined by their input.

This paper’s main complexity results are summarised in Table 1. The central insight here is that the least restricted settings yield an undecidable or non-elementary winning problem, and even for strong restrictions, the complexity

	No replay	Bounded	Unbounded
NWT	2-EXPTIME	undecidable	undecidable
ϵ -free NWT	co-NEXPTIME	non-elementary	undecidable
Relabelling	PSPACE	PSPACE	EXPTIME
Functional relabelling	NP	NP	PSPACE

Table 1: Summary of complexity results. All results are completeness results.

of the winning problem is generally quite high, with no tractable case among the standard settings. For this reason, we also study several limitations of these settings, derived from our lower bound proofs, in order to reduce complexity:

- *Depth-bounded NWT* lower the complexity of the replay-free case to EXPSPACE-complete (in comparison to 2-EXPTIME for general NWT).
- Strategies with *bounded Call width* lower the complexity of the bounded-replay case for ϵ -free NWT from non-elementary to co-NEXPTIME-complete or co-NP-complete, depending on the precise formalisation of bounded Call width.
- *Write-once* strategies yield a tractable case for functional relabelling transducers in a setting even more restrictive than the replay-free one.

Related Work Beyond the work already discussed, further complexity and decidability results for context-free games can be found in [2, 4], and [12] contains further references to related work.

Our concept of nested word transducers is based on Visibly Pushdown Transducers [13, 16], specifically the well-nested VPT of [8]. The original definitions of VPT in [13, 16] included ϵ -transitions, which were dropped from later definitions, as they caused several algorithmic problems, such as functionality and equivalence, to become undecidable (cf. [15]). Different from these approaches, this paper combines ϵ -transitions with the restriction to well-nested words, which is (to the best of the author’s knowledge) new research.

Organisation Section 2 gives basic notation and definitions. Section 3 defines nested word transducers and examines their structural and algorithmic properties. The next three sections give results on the complexity of the winning problem for games with transducer-based replacement, from most to least expressive – general nested word transducers (Section 4), nested word transducers without ϵ -transitions (Section 5), and relabelling transducers (Section 6). Each of these sections also discusses one of the restrictions with reduced complexity mentioned above. Section 7 concludes the paper. Due to space limitations, proofs and technical definitions are deferred to the appendix. The author is grateful to Gaetano Geck and Thomas Schwentick for careful proof-reading and valuable suggestions, and to the anonymous reviewers of MFCS 2016 for their insightful and constructive comments.

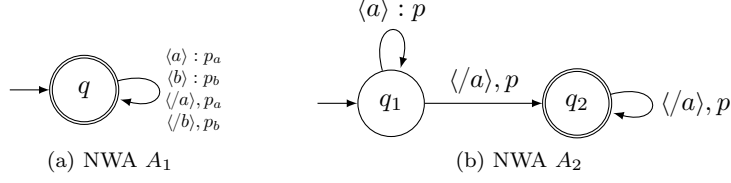


Figure 2: NWAs A_1 and A_2 from Example 1

2 Preliminaries

For any natural number $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$. For finite sets M , $\mathcal{P}(M)$ denotes the powerset of M , i.e. the set of all subsets of M . For an alphabet Σ , we denote the set of finite strings over Σ by Σ^* and ϵ denotes the empty string.

Nested words For a finite alphabet Σ , $\langle \Sigma \rangle \stackrel{\text{def}}{=} \{\langle a \rangle \mid a \in \Sigma\}$ denotes the set of all *opening* Σ -tags and $\langle \Sigma \rangle \stackrel{\text{def}}{=} \{\langle /a \rangle \mid a \in \Sigma\}$ the set of all *closing* Σ -tags. We denote by $\hat{\Sigma} \stackrel{\text{def}}{=} \langle \Sigma \rangle \cup \langle \Sigma \rangle$ the set of all Σ -tags. The set $\text{NW}(\Sigma) \subseteq \hat{\Sigma}^*$ of (well-)nested words (or (well-)nested strings) over Σ is the smallest set such that $\epsilon \in \text{NW}(\Sigma)$, and if $u, v \in \text{NW}(\Sigma)$ and $a \in \Sigma$, then also $u\langle a \rangle v\langle /a \rangle \in \text{NW}(\Sigma)$. We (informally) associate with every nested word w its *canonical forest representation*, such that words $\langle a \rangle\langle /a \rangle$, $\langle a \rangle v\langle /a \rangle$ and uv correspond to an a -labelled leaf, a tree with root a (and subforest corresponding to v), and the forest of u followed by the forest of v , respectively. A nested string w is *rooted* if its corresponding forest is a tree. We denote the set of rooted nested strings over Σ by $\text{rNW}(\Sigma)$. In a string $w = w_1 \dots w_n \in \hat{\Sigma}^*$, two tags $w_i \in \langle \Sigma \rangle$ and $w_j \in \langle \Sigma \rangle$ with $i < j$ are *associated* if the substring $w_i \dots w_j$ of w is a rooted nested string. An opening (closing) tag w_i in w is *unmatched*, if it has no associated closing (opening) tag in w . To stress the distinction from nested strings in $\text{NW}(\Sigma)$, we refer to strings in Σ^* as *flat strings*.

Nested word automata A *nested word automaton* (NWA) $A = (Q, P, \Sigma, \delta, q_0, F)$ [3] is basically a pushdown automaton which performs a push operation on every opening tag and a pop operation on every closing tag, and in which the pushdown symbols are just states. More formally, A consists of a set Q of *linear states*, a set P of *hierarchical states*, an alphabet Σ , a *transition relation* δ , an *initial state* $q_0 \in Q$, and a set $F \subseteq Q$ of *accepting (linear) states*. The relation δ is a subset of the union of sets $(Q \times \langle \Sigma \rangle \times Q \times P)$ and $(Q \times P \times \langle \Sigma \rangle \times Q)$. We sometimes interpret δ as the union of two functions from $(Q \times \langle \Sigma \rangle)$ to $\mathcal{P}(Q \times P)$ and from $(Q \times P \times \langle \Sigma \rangle)$ to $\mathcal{P}(Q)$ and write accordingly $(q', p) \in \delta(q, \langle a \rangle)$ for $(q, \langle a \rangle, q', p) \in \delta$ and $q' \in \delta(q, p, \langle /a \rangle)$ for $(q, p, \langle /a \rangle, q') \in \delta$. The semantics of NWA as well as the language $L(A)$ decided by a NWA A are defined in the natural way, with a NWA accepting if it reaches a configuration with an accepting state and empty stack. If A is a NWA, we call $L(A)$ a *regular language* (of nested words). A NWA is *deterministic* (or DNWA) if $|\delta(q, \langle a \rangle)| = 1 = |\delta(q, p, \langle /a \rangle)|$ for all $q \in Q$, $p \in P$ and $a \in \Sigma$. In this case, we simply write $\delta(q, \langle a \rangle) = (q', p')$ instead of $\delta(q, \langle a \rangle) = \{(q', p')\}$ (and accordingly for $\delta(q, p, \langle /a \rangle)$).

Example 1. The NWA A_1 (Fig. 2a) checks that its input string is well-nested by pushing hierarchical state p_a (resp. p_b) to the stack on each opening $\langle a \rangle$ (resp. $\langle b \rangle$) tag and popping an according hierarchical state with each matching closing tag. In this manner, A_1 decides the set of all well-nested strings over $\{a, b\}$. The NWA A_2 (Fig. 2b) initially pushes a hierarchical state p each time it reads $\langle a \rangle$ in linear state q_1 , then changes linear state to q_2 on reading the first $\langle /a \rangle$ and accepts iff each initial $\langle a \rangle$ is matched by a $\langle /a \rangle$. In this manner, it decides the language $\{\langle a \rangle^n \langle /a \rangle^n \mid n \geq 1\}$.

Context-free games A *context-free game (with transduction) on nested words* (cfG) $G = (\Sigma, \Gamma, R, T)$ consists of a finite alphabet Σ , a set $\Gamma \subseteq \Sigma$ of *function symbols*, a (replacement) *rule set* $R \subseteq \text{rNW}(\Sigma) \times \text{NW}(\Sigma)$ and a *target language* $T \subseteq \text{NW}(\Sigma)$. We will only consider the case where T is a non-empty regular nested word language and replacement rules are given by nested word transducers, to be defined in Section 3. A play of G is played by two players, JULIET and ROMEO, on a word $w \in \text{NW}(\Sigma)$. In a nutshell, JULIET moves the focus along w from left to right and decides for each closing tag $\langle /a \rangle$, whether she plays a *Read* or, in case $a \in \Gamma$, a *Call* move. In the latter case, ROMEO then replaces the rooted word u ending at the position of $\langle /a \rangle$ with some word v with $(u, v) \in R$ and the focus is set on the first symbol of v . If no such word v exists, ROMEO immediately wins the play. In case of a Read move, the focus just moves further on. JULIET wins a play if the word obtained at its end is in T .

Strategies A *strategy* for player $p \in \{J, R\}$ maps game states where player p is to move into allowed moves for player p , i.e. strategies σ for JULIET return moves in $\{\text{Read}, \text{Call}\}$ while strategies τ for ROMEO return replacement strings in $\text{NW}(\Sigma)$. Given an initial word w and strategies σ, τ the play $\Pi(\sigma, \tau, w)$ according to σ and τ on w is uniquely determined. A *winning strategy* for JULIET is a strategy σ such that JULIET wins the play $\Pi(\sigma, \tau, w)$, for every τ of ROMEO. By $\text{JWin}(G)$ we denote the set of all words for which JULIET has a winning strategy in G .

The *Call depth* of a play Π is the maximum nesting depth of Call moves in Π , if this maximum exists. That is, the Call depth of a play is zero, if no Call is played at all, and one, if no Call is played inside a string yielded by a replacement move. For a strategy σ of JULIET and a string $w \in \text{NW}(\Sigma)$, the *Call depth* $\text{Depth}^G(\sigma, w)$ of σ on w is the maximum Call depth in any play $\Pi(\sigma, \tau, w)$. A strategy σ has *k-bounded Call depth* if $\text{Depth}^G(\sigma, w) \leq k$ for all $w \in \text{NW}(\Sigma)$. As a more intuitive formulation, we use the concept of *replay*: Strategies for JULIET of Call depth one are called *replay-free*, and strategies of *k-bounded Call depth*, for any k , have *bounded replay*.

Algorithmic problems In this paper, we study the following algorithmic problem $\text{JWin}(\mathcal{G})$ for various classes \mathcal{G} of context-free games with replacement transducers.

$\text{JWin}(\mathcal{G})$	
Given:	A context-free game $G \in \mathcal{G}$ and a string w .
Question:	Is $w \in \text{JWin}(G)$?

A class \mathcal{G} of context-free games in $\text{JWin}(\mathcal{G})$ generally comes with three parameters:

- the representation of the target language T ,
- the representation of the replacement relation R , and
- to which extent replay is restricted.

We generally assume target languages to be represented by DNWAs, because the complexity of the winning problem is already quite high under that assumption, and our main interest is in finding classes \mathcal{G} for which $\text{JWIN}(\mathcal{G})$ is tractable. Replacement relations will be given as different types of nested word transducers (defined in Section 3). By a slight abuse of notation, the replacement transducer implementing a replacement relation R will also be referred to as R .

In each setting, we consider the cases of unrestricted replay, bounded replay (Call depth k , for some k), and no replay (Call depth 1). We note that replay depth is formally not an actual game parameter, but the algorithmic problem can be restricted to strategies of JULIET of the stated kind. If the class \mathcal{G} of games is clear from the context, we often simply write JWIN instead of $\text{JWIN}(\mathcal{G})$.

3 Nested Word Transducers

In this section, we define nested word transducers and examine their closure properties and complexities of algorithmic problems. Thanks to our definition putting some rather severe restrictions on the use of ϵ -transitions and the allowed output of transducers, we obtain advantageous closure properties and comparatively low complexities.

Intuitively, a NWT T works much like a NWA with output and additional ϵ -transitions – T reads its input from left to right and decides nondeterministically which available transition to use; on an opening (resp. closing) transition, it reads an opening (closing) input tag, changes its linear state and pushes (pops) a hierarchical state while producing an output. Opening (closing, internal) ϵ -transitions do not consume input symbols but induce state changes and outputs. T only produces an output string if it accepts the input string.

Definition 2. A *nested word transducer* (or *NWT*) is a tuple $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ consisting of a set Q of *linear states*, a set P of *hierarchical states*, a set $P_\epsilon \subseteq P$ of *hierarchical ϵ -states*, an alphabet Σ , a *transition relation* δ , which is the union of three relations from $(Q \times (\langle \Sigma \rangle \cup \{ \langle \epsilon \rangle \}) \times Q \times P \times \hat{\Sigma}^*)$ (called *opening transitions*), $(Q \times \{ \epsilon \} \times Q \times \text{NW}(\Sigma))$ (called *internal transitions*) and $(Q \times P \times (\langle \Sigma \rangle \cup \{ \langle \epsilon \rangle \}) \times Q \times \hat{\Sigma}^*)$ (called *closing transitions*), an *initial state* $q_0 \in Q$, and a set of *accepting states* $F \subseteq Q$, such that for all $q, q', r, r' \in Q$, $p \in P$, $a \in \Sigma \cup \{ \epsilon \}$ and $u, v \in \hat{\Sigma}^*$ it holds that¹

- $(q, \langle \epsilon \rangle, q', p, u) \in \delta$ or $(q, p, \langle \epsilon \rangle, q', u) \in \delta$ if and only if $p \in P_\epsilon$ (*ϵ -consistency*),
- if $(q, \langle a \rangle, q', p, u) \in \delta$ and $(r, p, \langle /a \rangle, r', v) \in \delta$, then $uv \in \text{NW}(\Sigma)$ (*well-formedness*), and

¹These three conditions make NWT roughly correspond to *synchronized visibly pushdown transducers* [13]; we mainly require them to ensure closure of regular nested word languages under NWT transduction.

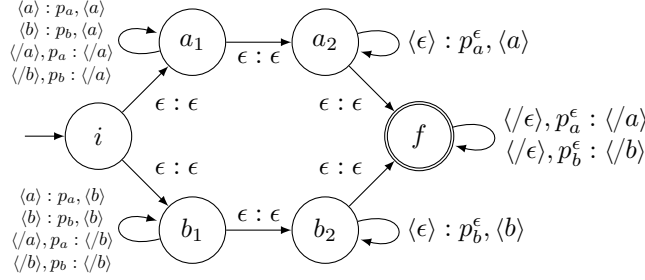


Figure 3: Nested Word Transducer T_{ab} from Example 3.

- for each $(q, \langle a \rangle, q', p, u) \in \delta$ (resp. $(r, p, \langle /a \rangle, r', u) \in \delta$) with $u \neq \epsilon$, u contains at least one unmatched opening (resp. closing) tag (*synchronization*).

As for standard NWA, we also write $(q', p, u) \in \delta(q, \langle a \rangle)$ (resp. $(q', u) \in \delta(q, p, \langle /a \rangle)$), $(q', u) \in \delta(q, \epsilon)$ instead of $(q, \langle a \rangle, q', p, u) \in \delta$ (resp. $(q, p, \langle /a \rangle, q', u), (q, \epsilon, q', u) \in \delta$).

A detailed semantics definition can be found in the appendix.

Example 3. Figure 3 shows a NWT T_{ab} , with linear states displayed as circles and transitions as arrows. From the initial state i , T_{ab} branches nondeterministically into either state a_1 or b_1 . In state a_1 , T_{ab} checks that the input string is well-nested just as the NWA A_1 from Example 1. During this check, T_{ab} outputs $\langle a \rangle$ (resp. $\langle /a \rangle$) for each opening (resp. closing) input tag, effectively relabelling the input string to consist exclusively of a -labelled tags. In state a_2 , T_{ab} inserts into the output string an arbitrary number of opening $\langle a \rangle$ tags, for which a matching number of $\langle /a \rangle$ tags are inserted in state f before T_{ab} accepts. The behaviour of T_{ab} in states b_1 and b_2 is analogous, but outputs consist only of b -labelled tags. Altogether, T_{ab} chooses nondeterministically some $x \in \{a, b\}$, relabels all tags of a well-nested input string into x -labelled tags and then appends a string of the form $\langle x \rangle^n \langle /x \rangle^n$.

The *image* $T(w)$ of a well-nested string $w \in \text{NW}(\Sigma)$ under T is the set of all outputs of T on w according to some accepting run of T on w . This definition extends to sets of input strings in the natural way: For a set $S \subseteq \text{NW}(\Sigma)$, we define $T(S) = \bigcup_{w \in S} T(w)$. The *domain* $\mathcal{D}(T)$ of T is the set of all strings w such that $T(w) \neq \emptyset$, and the *range* $\mathcal{R}(T)$ of T is the set of all strings u such that there exists a $w \in \text{NW}(\Sigma)$ with $u \in T(w)$, i.e. the set of all possible outputs of T .

We next define several restrictions on the expressiveness of NWT.

Definition 4. Let $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ be a NWT. We call T

- *ϵ -free* if $P_\epsilon = \emptyset$ and δ contains no ϵ -transitions.
- *non-deleting* if the output component of every non-internal transition in δ is a non-empty string;
- *deterministic* (or a DNWT) if for every $q \in Q, p \in P$ and $a \in \Sigma$, it holds that $|\delta(q, \langle a \rangle)| = |\delta(q, p, \langle /a \rangle)| = 1$;

- a *relabelling* transducer if it is ϵ -free and for every $q, q' \in Q, p \in P, a \in \Sigma$ and $u \in \Sigma^*$, if $(q', p, u) \in \delta(q, \langle a \rangle)$, then $u \in \langle \Sigma \rangle$, and if $(q', u) \in \delta(q, p, \langle /a \rangle)$, then $u \in \langle / \Sigma \rangle$;
- *functional*, if for every $w \in \text{NW}(\Sigma)$, it holds that $|T(w)| = 1$.

It is easy to see that the length of any output of an ϵ -free NWT is at most linear in the length of the input string, while outputs of general NWT may grow to an arbitrary length. We note that functionality, unlike the other restrictions defined here, is a *semantic* condition. We do not investigate in this paper the decidability or complexity of determining whether a NWT is functional; likely, techniques for Visibly Pushdown Transducers in [8] could be adapted for this purpose. Also, while determinism implies functionality, the converse does not hold.

The following lemma shows that we can assume without loss of generality that each transition of a NWT involves at most one input and at most one output symbol, i.e. each opening (closing) transition outputs at most one opening (closing) tag and each internal ϵ -transition outputs nothing.

Lemma 5. Each NWT $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ can be transformed in polynomial time into an NWT $T' = (Q', P', P'_\epsilon, \Sigma, \delta', q_0, F)$ with $T(w) = T'(w)$ for each $w \in \text{NW}(\Sigma)$, such that for any transition in δ' with output u , it holds that $|u| \leq 1$.

We say that a NWT of this shape is in *normal form*.

In most of this paper, we restrict our attention to non-deleting transducers. This is because regular nested word languages are closed under transduction by non-deleting NWT, which does not hold in the presence of deletions (consider, for instance, a NWT deleting all matched opening and closing c -labelled tags on the regular input language $\{(\langle a \rangle \langle /a \rangle \langle c \rangle)^n (\langle /c \rangle \langle b \rangle \langle /b \rangle)^n \mid n \geq 0\}$). The practical motivation for desiring this property is the fact that the AXML setting assumes that function call results can be specified by standard XML schema languages, which are subclasses of regular nested word languages.

Moreover, for most of the transducer models examined here, non-deleting transducers are not a significant restriction when it comes to context-free games, as the following result states.

Lemma 6. Any context-free game $G = (\Sigma, \Gamma, R, T)$ with NWT R can be transformed in polynomial time into a game $G' = (\Sigma', \Gamma, R', T')$ such that R' is non-deleting and it holds that $\text{JWin}(G') \cap \text{NW}(\Sigma) = \text{JWin}(G)$.

Using Lemma 5, it is comparatively easy (if tedious) to prove that non-deleting NWTs are closed under composition. This proof, like most proofs for properties of NWT in this section, follows proof ideas used in [8, 13] adapted to the specifics of NWT.

Proposition 7. Let T_1, T_2 be non-deleting NWT. Then there exists a non-deleting NWT T such that for all $w \in \text{NW}(\Sigma)$, it holds that $T(w) = (T_2 \circ T_1)(w) \stackrel{\text{def}}{=} T_2(T_1(w))$. This NWT T can be computed from T_1 and T_2 in polynomial time and is of size $\mathcal{O}(|T_1| \cdot |T_2|)$.

Since we are solely interested in NWTs operating on well-nested strings, we restrict our attention to NWTs with well-nested domains. The following corollary to Proposition 7 justifies this restriction.

Corollary 8. Let T be a non-deleting NWT and A a NWA over alphabet Σ . Then, there exists a non-deleting NWT T' of size $\mathcal{O}(|T| \cdot |A|)$ such that $\mathcal{D}(T') = \mathcal{D}(T) \cap L(A)$ and $T'(w) = T(w)$ for each $w \in \mathcal{D}(T) \cap L(A)$.

In order to prove closure of regular nested word languages under transduction by non-deleting NWT, we observe another helpful property of these transducers.

Lemma 9. Let T be a non-deleting NWT with $\mathcal{D}(T) \subseteq \text{NW}(\Sigma)$. Then $\mathcal{R}(T)$ is a regular language of nested words.

Corollary 10. Regular nested word languages are closed under transduction by non-deleting NWT, i.e. if $L \subseteq \text{NW}(\Sigma)$ is regular and T an NWT, then $T(L)$ is regular.

We now turn to the complexity of standard decision problems for NWT. The upper bounds use relatively simple constructions based on Proposition 7, while lower bounds follow from comparable results for NWA.

Theorem 11. The membership problem for non-deleting NWT (Given a non-deleting NWT T and strings $w, u \in \text{NW}(\Sigma)$, is $u \in T(w)$?) is in PTIME.

Theorem 12. The nonemptiness problem for non-deleting NWT (Given a non-deleting NWT T , is there a string $w \in \text{NW}(\Sigma)$ with $T(w) \neq \emptyset$?) is PTIME-complete with regard to logspace reductions.

Theorem 13. The type checking problem for non-deleting NWT (Given a non-deleting NWT T and NWA A_1, A_2 , is $T(L(A_1)) \subseteq L(A_2)$?) is

- (a) EXPTIME-complete in general, and
- (b) PTIME-complete (w.r.t. logspace reductions) if A_2 is a DNWA.

4 Games with general NWT replacement

Having laid the foundation with basic results on NWT, we now examine context-free games with NWT-based replacement. The main characteristic distinguishing general NWT from ϵ -free NWT is the fact that, for any input string w and NWT T , transducts in $T(w)$ may be arbitrarily large in the size of w . This behaviour is necessary if we want to simulate games with regular replacement languages (in the sense of [14]) by transducer-based games. As it turns out, however, NWT-based replacement is much more complex than that: the winning problem in games with replay becomes undecidable (as opposed to 2-EXPTIME with regular replacement languages), which may be proven by a rather straightforward reduction from the complement of the halting problem for Turing machines.

Theorem 14. For the class of games with NWT and Call depth $k \geq 2$, JWIN is not recursively enumerable.

Even the replay-free winning problem for JULIET is quite hard when using NWT for replacement – we show that this problem is complete for doubly exponential time. The lower bound uses a rather intricate reduction from a two-player tiling problem, while the upper bound is proven by reduction to the purely NWT-based problem of *alternating iterated transduction*, which can be proven to be in 2-EXPTIME.

Theorem 15. For the class of replay-free games with NWT, JWIN is 2-EXPTIME-complete.

The lower bound proofs for both of these results require replacement transducers to output nested words of arbitrary depth. Considering our practical motivation, it is rarely required that function calls in Active XML documents return arbitrarily deep trees. Therefore, we now investigate the impact of limiting replacement transducers' output depth.

For simplicity's sake, we assume depth-boundedness as a *semantic* restriction, i.e. we assert that all outputs in $R(w)$ produced by a depth-bounded replacement transducer R on a string w obey a given upper bound on their depth, without examining the decidability and complexity of determining whether or not a given transducer is depth-bounded.

We note that NWT with an output depth linear in the size of the input string are already strictly more expressive than ϵ -free NWT, so the lower bounds from Section 5 also hold for NWT with linear output depth. As these lower bounds are already quite high, we focus only on transducers whose output depth is bounded by a constant.

Definition 16. An NWT R is called *depth-bounded* if there is some constant $d \geq 0$ such that for any $w \in \text{NW}(\Sigma)$ and any $w' \in R(w)$, the depth of w' is at most d .

Using depth-bounded NWT as replacement transducers places the complexity of the winning problem between those for general NWT and for ϵ -free NWT. The upper and lower bounds are proven similarly to those of Theorem 15, but use the fact that the stack size of a depth-bounded NWT on a fixed input is bounded by a constant.

Theorem 17. For the class of replay-free games with depth-bounded NWT, JWIN is EXPSPACE-complete.

5 Games with ϵ -free NWT replacement

In this section, we examine context-free games with replacement relations given by ϵ -free NWT. As we shall see, this leads to a decidable winning problem for games with bounded replay, but non-elementary complexity in all but the easiest case. For the unbounded replay case, we can construct a rather straightforward reduction from the halting problem for TMs.

Theorem 18. For the class of games with ϵ -free NWT and unbounded replay, JWIN is undecidable.

Different from games with general NWT, the winning problem for JULIET in games with ϵ -free NWT and fixed Call depth is decidable; however, the complexity of deciding JWIN is already non-elementary for Call depth 2.

Theorem 19. For the class of games with ϵ -free NWT and Call depth bounded by $d \geq 2$, JWIN is decidable, but not decidable in elementary time.

Even for replay-free games with ϵ -free NWT, the complexity of deciding the winning problem for JULIET is still rather high. The lower bound is proven by

reduction from a tiling problem, and the CO-NEXPTIME algorithm uses non-determinism to guess moves for ROMEO while trying out all possible strategies for JULIET by backtracking.

Theorem 20. For the class of replay-free games with ϵ -free NWT, JWIN is complete for CO-NEXPTIME.

The non-elementary lower bound in Theorem 19 follows from the fact that, in each string returned by ROMEO, JULIET may play Call arbitrarily often. On a return string corresponding to a path of length n , JULIET may play Call on all n nodes bottom-up, with each such Call doubling the number of nodes below the called node, inducing a non-elementary blow-up.

To avoid this, we now examine games with bounded *Call width*, where, intuitively, JULIET may only play Call for a bounded number of times in each replacement string given by ROMEO. Note that Call width is counted within each individual replacement string – so, in a game of Call depth 3 and Call width c , if JULIET plays Call on some position of the input string, she may then place up to c calls within the string returned by ROMEO, and again up to c calls in *each* of the depth-2 replacement strings resulting from those calls.

More formally, the Call width of a play Π is the maximum number of times JULIET plays Call in any replacement string given by ROMEO in Π . This definition extends naturally into that of Call width of a strategy. Note that Call width only applies to *replacement* strings, so JULIET may still call arbitrarily many positions of the *input* string, even for games with Call width 0. For this reason, replay-free strategies always have bounded Call width.

The proof of Theorem 18 shows that JWIN remains undecidable for games with unbounded Call depth, even with Call width bounded by 1. For bounded replay, though, the complexity of JWIN collapses to that of the replay-free case if Call width is bounded.

Theorem 21. For the class of games with ϵ -free NWT, Call depth bounded by $d \geq 1$ and Call width bounded by $k \geq 1$, JWIN is CO-NEXPTIME-complete.

As mentioned above, bounded Call width does not affect JULIET's options for Call moves on the input string, as we generally want JULIET to be able to at least process *all* function symbols in the input. Dropping this requirement (i.e. bounding *Call width including input*) yields at least an exponential improvement in complexity.

Theorem 22. For the class of games with ϵ -free NWT, Call depth bounded by d and Call width including input bounded by k , JWIN is

- (a) CO-NP-complete for $d \geq 1$ and $k \geq 2$,
- (b) CO-NP-complete for $d \geq 2$ and $k \geq 1$, and
- (c) in PTIME for $d = k = 1$.

The upper bounds in Theorems 21 and 22 use a backtracking algorithm like the one for Theorem 20; in this case, however, bounded Call width reduces the size of both the decision tree for JULIET and the occurring replacement strings.

6 Games with relabelling replacement

As seen before, even the limited amount of insertion allowed by ϵ -free NWT renders the winning problem for JULIET quite complex. We now examine how this changes if we disallow insertion entirely. First, we show that the winning problem is greatly simplified by the fact that transducts of relabelling transducers do not require any additional space beyond that provided by the input. In fact, the upper bounds of Theorems 23 to 26 all use a nigh-trivial (alternating or nondeterministic) algorithm that simply simulates the game. Lower bounds, on the other hand, are proven by reduction from the word problem for linearly bounded (alternating) Turing machines (Theorems 23 and 25) and from standard logic-based problems (Theorems 24 and 26).

Theorem 23. For the class of games with relabelling transducers and unbounded replay, JWIN is EXPTIME-complete.

With limited or no replay, the complexity decreases even further.

Theorem 24. For any $k \geq 1$, for the class of games with relabelling transducers and bounded Call depth k , JWIN is PSPACE-complete.

As the winning problem for JULIET remains intractable (assuming $\text{PTIME} \neq \text{PSPACE}$) for replay-free games with relabelling transducers, we now turn to the even more limited class of functional relabellings. Note that games with functional transducers are essentially “solitaire games” for JULIET, as they do not allow for any choice of transducts by ROMEO.

Theorem 25. For the class of games with functional relabelling transducers and unbounded replay, JWIN is PSPACE-complete.

As for general relabelling transducers, the complexity of JWIN is the same for games with bounded replay and no replay when restricted to functional relabelling transducers.

Theorem 26. For any $k \geq 1$, for the class of games with functional relabelling transducers and bounded Call depth k , JWIN is NP-complete.

We see that even in this very simple class of games, we still fail to obtain a PTIME upper bound. Careful examination of lower bound proofs shows that our semantics for replay-free games still allows for a sort of “hidden replay”: On a string of the form $\langle a \rangle \langle b \rangle v \langle /b \rangle \langle /a \rangle$, if JULIET plays Call first on $\langle /b \rangle$ then on $\langle /a \rangle$, the substring v undergoes *two* transductions – one from the Call to $\langle /b \rangle$, another from the Call to $\langle /a \rangle$. This allows us to perform any number d of transductions on a given string by enclosing it inside d nested function symbols.

Excluding this hidden replay yields a very narrow restriction of context-free games, which we call *write-once* games. In these, no substring may be transduced more than once, i.e. JULIET may only play Call on any closing tag $\langle /a \rangle$ if the substring enclosed in it does not contain a substring on which JULIET has played Call before. Note that write-once games are always replay-free, but even weaker as far as JULIET’s rewriting capabilities are concerned.

A slight adaptation of the proof of Theorem 24 shows that JWIN remains PSPACE-hard for write-once games with arbitrary relabelling transducers; for functional (and deterministic) relabelling transducers, however, we can prove

tractability. The proof constructs from a given game G a NWT R_J such that for each $w \in \text{NW}(\Sigma)$, the set of all strings into which JULIET may rewrite w in G is given by $R_J(w)$.

Theorem 27. For the class of write-once games with functional relabelling transducers, JWIN is in PTIME.

7 Conclusion

The research presented in this paper shows that a major challenge in using transducers for context-free games is finding sensible transducer models and strategy restrictions that do not cause a prohibitive increase in the complexity of the winning problem compared to context-free games without parameter transformation. This paper has made a first step towards identifying what suitable restrictions may look like; however, the few tractable cases identified here are still so restricted that they may be only of limited practical interest.

It is possible that the complexity of the winning problem in games with replacement transducers may be further reduced by restricting relevant schemas to be closer to practical schema specifications for XML (such as DTDs or XML Schema). However, since research in [14] indicates that specifications of input schemas for external services influence the complexity of the safe rewriting problem, further research might be necessary to find transducer models whose input and output schemas can be described by DTDs or XML Schema.

References

- [1] Serge Abiteboul, Omar Benjelloun, and Tova Milo. The Active XML project: an overview. *VLDB J.*, 17(5):1019–1040, 2008.
- [2] Serge Abiteboul, Tova Milo, and Omar Benjelloun. Regular rewriting of active XML and unambiguity. In *PODS*, pages 295–303, 2005.
- [3] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.
- [4] Henrik Björklund, Martin Schuster, Thomas Schwentick, and Joscha Kulbatzki. On optimum left-to-right strategies for active context-free games. In *ICDT*, pages 105–116, 2013.
- [5] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.
- [6] B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.
- [7] Stephen A. Cook. A hierarchy for nondeterministic time complexity. *J. Comput. Syst. Sci.*, 7(4):343–353, 1973.
- [8] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of visibly pushdown transducers. In *MFCS*, pages 355–367, 2010.

- [9] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*. Springer, 2002.
- [10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - (2. ed.)*. Addison-Wesley series in computer science. Addison-Wesley-Longman, 2001.
- [11] Tova Milo, Serge Abiteboul, Bernd Amann, Omar Benjelloun, and Frédéric Dang Ngoc. Exchanging intensional XML data. *ACM Trans. Database Syst.*, 30(1):1–40, 2005.
- [12] Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory Comput. Syst.*, 39(1):237–276, 2006.
- [13] Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *ICALP (2)*, pages 386–397, 2008.
- [14] Martin Schuster and Thomas Schwentick. Games for Active XML revisited. In *ICDT*, pages 60–75, 2015.
- [15] Frédéric Servais. *Visibly pushdown transducers*. Dissertation, ULB Belgique, 2011.
- [16] Alex Thomo, S. Venkatesh, and Ying Ying Ye. Visibly pushdown transducers for approximate validation of streaming xml. In *FoIKS*.
- [17] Margus Veanes. On computational complexity of basic decision problems of finite tree automata. Technical report, UPMail Technical Report 133, Uppsala University, Computing Science Department, 1997.

A Appendix

For easier reference, we restate the results that were already stated in the body of the paper. Definitions and results not stated in the body can be identified by their number of the type A.xxx.

Additional preliminaries

This section gives additional notations and definitions needed only for proofs in the following sections.

Nested words For the purpose of reductions, we will sometimes need to encode flat strings as nested strings; for a string $w = w_1 \dots w_n \in \Sigma^*$ with $w_1, \dots, w_n \in \Sigma$, the *standard nested string encoding* \hat{w} of w is $\hat{w} = \langle w_1 \rangle \langle /w_1 \rangle \dots \langle w_n \rangle \langle /w_n \rangle \in \text{NW}(\Sigma)$.

Nested word automata A *configuration* κ of A is a tuple $(q, \alpha) \in Q \times P^*$, with a linear state q and a sequence α of hierarchical states, reflecting the pushdown store. A *run of A on $w = w_1 \dots w_n \in \text{NW}(\Sigma)$* is a sequence $\kappa_0, \dots, \kappa_n$ of configurations $\kappa_i = (q_i, \alpha_i)$ of A such that for each $i \in [n]$ and $a \in \Sigma$ it holds that κ_i is a *successor configuration of κ_{i-1} with w_i* , i.e. that

- if $w_i = \langle a \rangle$, then $(q_{i-1}, \langle a \rangle, q_i, p) \in \delta$ (for some $p \in P$), and $\alpha_i = p\alpha_{i-1}$, or
- if $w_i = \langle /a \rangle$, then $(q_{i-1}, p, \langle /a \rangle, q_i) \in \delta$ (for some $p \in Q$), and $p\alpha_i = \alpha_{i-1}$.

In this case, we also write $\kappa_0 \xrightarrow{w}_A \kappa_n$. We say that A *accepts w* if $(q_0, \epsilon) \xrightarrow{w}_A (q', \epsilon)$ for some $q' \in F$. The language $L(A) \subseteq \text{NW}(\Sigma)$ is defined as the set of all strings accepted by A .

Context-free games Towards a formal definition of play, a *configuration* is a tuple $\kappa = (p, u, v) \in \{J, R\} \times \hat{\Sigma}^* \times \hat{\Sigma}^*$ where p is the player to move, $uv \in \text{NW}(\Sigma)$ is the *current word*, and the first symbol of v is the *current position*. A *winning configuration for JULIET* is a configuration $\kappa_J = (J, u, \epsilon)$ with $u \in T$, and a *winning configuration for ROMEO* is a configuration $\kappa_R = (R, u_1 \langle a \rangle v, \langle /a \rangle u_2)$, with $\langle a \rangle v \langle /a \rangle \in \text{rNW}(\Sigma)$ such that there is no $v' \in \text{NW}(\Sigma)$ with $(\langle a \rangle v \langle /a \rangle, v') \in R$. The configuration $\kappa' = (p', u', v')$ is a *successor configuration of $\kappa = (p, u, v)$* (Notation: $\kappa \rightarrow \kappa'$) if one of the following holds:

- (1) $p' = p = J$, $u' = us$, and $sv' = v$ for some $s \in \hat{\Sigma}$ (JULIET plays Read);
- (2) $p = J$, $p' = R$, $u = u'$, $v = v' = \langle /a \rangle z$ for $z \in \hat{\Sigma}^*$, $a \in \Gamma$, (JULIET plays Call);
- (3) $p = R$, $p' = J$, $u = x \langle a \rangle y$, $v = \langle /a \rangle z$ for $x, z \in \hat{\Sigma}^*$, $\langle a \rangle y \langle /a \rangle \in \text{rNW}(\Sigma)$, $u' = x$ and $v' = y'z$ for some $y' \in \text{NW}(\Sigma)$ with $(\langle a \rangle y \langle /a \rangle, y') \in R$ (ROMEO plays y').

The *initial configuration* of game G for string w is $\kappa_0(w) \stackrel{\text{def}}{=} (J, \epsilon, w)$. A *play* of G is either an infinite sequence $\Pi = \kappa_0, \kappa_1, \dots$ or a finite sequence $\Pi = \kappa_0, \kappa_1, \dots, \kappa_k$ of configurations, where, for each $i > 0$, $\kappa_{i-1} \rightarrow \kappa_i$ and, in the finite case, κ_k has no successor configuration. In the latter case, JULIET *wins* the play if κ_k is a winning position for JULIET, in all other cases, ROMEO wins.

Strategies A *strategy* for player $p \in \{J, R\}$ maps prefixes $\kappa_0, \kappa_1, \dots, \kappa_k$ of plays, where κ_k is a p -configuration, to allowed moves. We denote strategies for JULIET by $\sigma, \sigma', \sigma_1, \dots$ and strategies for ROMEO by $\tau, \tau', \tau_1, \dots$.

A strategy σ is *memoryless* if, for every prefix $\kappa_0, \kappa_1, \dots, \kappa_k$ of a play, the selected move $\sigma(\kappa_0, \kappa_1, \dots, \kappa_k)$ only depends on κ_k . As context-free games are reachability games we only need to consider memoryless strategies; see, e.g., [9].

Proposition A.1. Let G be a context-free game, and w a string. Then either JULIET or ROMEO has a winning strategy on w , which is actually memoryless.

Therefore, in the following, strategies σ for JULIET map configurations κ to moves $\sigma(\kappa) \in \{\text{Call}, \text{Read}\}$ and strategies τ for ROMEO map configurations κ to moves $\tau(\kappa) \in \text{NW}(\Sigma)$.

For configurations κ, κ' and strategies σ, τ we write $\kappa \xrightarrow{\sigma, \tau} \kappa'$ if κ' is the unique successor configuration of κ determined by strategies σ and τ . Given an initial word w and strategies σ, τ the play² $\Pi(\sigma, \tau, w) \stackrel{\text{def}}{=} \kappa_0(w) \xrightarrow{\sigma, \tau} \kappa_1 \xrightarrow{\sigma, \tau} \dots$ is uniquely determined. If $\Pi(\sigma, \tau, w)$ is finite, we denote the word represented by its final configuration by $\text{word}_G(w, \sigma, \tau)$.

A strategy σ for JULIET is *finite* on string w if the play $\Pi(\sigma, \tau, w)$ is finite for every strategy τ of ROMEO. It is a *winning strategy* on w if JULIET wins the play $\Pi(\sigma, \tau, w)$, for every τ of ROMEO. A strategy τ for ROMEO is a *winning strategy* for w if ROMEO wins $\Pi(\sigma, \tau, w)$, for every strategy σ of JULIET. We only consider finite strategies for JULIET, due to JULIET's winning condition. We denote the set of all finite strategies for JULIET in the game G by $\text{STRAT}_J(G)$, and the set of all strategies for ROMEO by $\text{STRAT}_R(G)$.

Tilings Several of our lower bound proofs use algorithmic problems involving tilings. We will define the individual algorithmic problems in later sections as needed and only give basic common definitions here.

A *tiling of height m and width ℓ* (or $m \times \ell$ -tiling) over a tile set U with vertical constraints $V \subseteq U \times U$, horizontal constraints $H \subseteq U \times U$, initial tile $u_i \in U$ and final tile $u_f \in U$ is a mapping $t : [m] \times [\ell] \rightarrow U$ such that

- $(t(i, j), t(i + 1, j)) \in V$ for each $i \in [m - 1]$ and $j \in [\ell]$,
- $(t(i, j), t(i, j + 1)) \in H$ for each $i \in [m]$ and $j \in [\ell - 1]$,
- $t(1, 1) = u_i$, and
- $t(m, \ell) = u_f$.

Intuitively, a tiling arranges $m \cdot \ell$ tiles from U in m rows and ℓ columns such that the first row starts with the initial tile, the last row ends with the final tile and horizontally or vertically adjacent tiles match (as per the horizontal and vertical constraints). Accordingly, for a tiling t of width ℓ and height m , we refer to the string $t(i, 1)t(i, 2) \dots t(i, \ell) \in U^*$ as the i -th row and to $t(1, j)t(2, j) \dots t(m, j) \in U^*$ as the j -th column of t .

We will often encode an $m \times \ell$ -tiling t as a string w_t of the form $(U^\ell \#)^m$ using a special *line divider symbol* $\# \notin U$, with the interpretation that $t(i, j)$ is the $(i - 1) \cdot (\ell + 1) + m$ -th symbol of w_t .

²As the underlying game G will always be clear from the context, our notation does not mention G explicitly.

Proofs and additional results for Section 3

The following result is used in several complexity proofs for Section 3.

Theorem A.2. The emptiness problem for DNWA (given a DNWA A , is $L(A) = \emptyset$?) is complete for PTIME with regard to logspace reductions.

Proof. The upper bound was proven in [3]. The lower bound can be proven by a straightforward (if technical) reduction from the emptiness problem for deterministic top-down tree automata (cf. Section 7.2 in [3] and Theorem 1 in [17]). \square

In the next subsection, we extend nested word automata by (internal) ϵ -transitions. These ϵ -NWA will be of use in the proofs for section 3.

Nested Word Automata with ϵ -transitions

It is well known (see, for instance, [10]) that extending finite-state automata with ϵ -transitions does not change their expressive power; (nondeterministic) finite-state automata with ϵ -transitions still decide exactly the class of regular languages of flat strings.

Even though nested word automata and the class of regular nested word languages strongly parallel finite-state automata and flat regular languages, a similar investigation has so far not been performed for nested word automata. We now define and examine nested word automata with ϵ -transitions, mainly as a tool for the analysis of nested word transducers with ϵ -transitions.

Definition A.3. A *Nested Word Automaton with ϵ -transitions* (ϵ -NWA) $A = (Q, P, \Sigma, \delta, q_0, F)$ consists of

- a set Q of *linear states*,
- a set P of *hierarchical states*,
- an alphabet Σ ,
- a *transition relation* $\delta \subseteq (Q \times \langle \Sigma \rangle \times Q \times P) \cup (Q \times P \times \langle / \Sigma \rangle \times Q) \cup (Q \times \{\epsilon\} \times Q)$,
- an *initial state* $q_0 \in Q$, and
- a set of *accepting states* $F \subseteq Q$.

As for standard NWA, we also write $(q', p) \in \delta(q, \langle a \rangle)$ (resp. $q' \in \delta(q, p, \langle /a \rangle)$, $q' \in \delta(q, \epsilon)$) instead of $(q, \langle a \rangle, q', p) \in \delta$ (resp. $(q, p, \langle /a \rangle, q')$, $(q, \epsilon, q') \in \delta$).

Note that ϵ -transitions are always *internal* transitions that merely change the current linear state, not the hierarchical stack; allowing ϵ -transitions to manipulate the stack as well would yield a strictly more expressive automaton model.

The semantics of ϵ -NWA is defined almost exactly like that of NWA, by way of (accepting) runs. The only difference is that for a run $\kappa_0, \dots, \kappa_n$ of an ϵ -NWA $A = (Q, P, \Sigma, \delta, q_0, F)$ on a nested string $w \in \text{NW}(\Sigma)$, it merely holds that $n \geq |w|$, and each configuration κ_i is either a successor configuration of κ_{i-1} with the next unread symbol of w (as defined in Section 2), or a successor configuration with ϵ , i.e. $\kappa_{i-1} = (q, \alpha)$, $\kappa_i = (q', \alpha)$ and $(q, \epsilon, q') \in \delta$.

The following properties of ϵ -NWA follow easily from the proofs for corresponding properties of NWA in [3].

Lemma A.4. For each ϵ -NWA A , there exists a DNWA A' of size at most exponential in $|A|$ such that $L(A) = L(A')$.

Proof. (Sketch) The proof of this statement for NWA without ϵ -transitions in Theorem 3.3 of [3] uses a modified subset construction, where states of the exponential-sized DNWA A' correspond to sets of pairs of states of A (so-called *summaries*) such that if a (not necessarily well-nested) string w induces a (partial) run from the initial state of A' to some summary state $S \in \mathcal{P}(Q \times Q)$, then there are (partial) runs with w from each $q \in Q$ to all $q' \in \{q' \mid (q, q') \in S\}$ in A' .

To account for ϵ -transitions, we modify these summaries to include ϵ -closures of target states, i.e. for each pair (q, q') of states contained in a summary S as constructed in [3], we add to S all pairs (q, q'') , where q'' is reachable from q by a series of ϵ -transitions in A . Otherwise, the construction (and correctness proof) is the same as in [3]. \square

Lemma A.5. For all ϵ -NWA A_1 and A_2 , it is possible to construct in polynomial time ϵ -NWA deciding $L(A_1) \cup L(A_2)$ and $L(A_1) \cap L(A_2)$.

Proof. (Sketch) This proof, like the one for Theorem 3.5 in [3], uses a standard product construction simulating A_1 and A_2 simultaneously on the input. The product automaton $A' = (Q_1 \times Q_2, P_1 \times P_2, \Sigma, \delta', (q_{0,1}, q_{0,2}), F')$ is constructed as in [3] and simply extended by ϵ -transitions. Note that, unlike reading transitions, ϵ -transitions do *not* have to be synchronised between the two automata, i.e. an ϵ -transition of A' simulates an ϵ -transition of only *one* of the component automata A_1 or A_2 . The transition relation δ' of A' is therefore extended by the sets $\{((q_1, q_2), \epsilon, (q'_1, q_2)) \mid (q_1, \epsilon, q'_1) \in \delta_1, q_2 \in Q_2\}$ and $\{((q_1, q_2), \epsilon, (q_1, q'_2)) \mid (q_2, \epsilon, q'_2) \in \delta_2, q_1 \in Q_1\}$. \square

Theorem A.6. (a) The membership and emptiness problem for ϵ -NWA are in PTIME.

(b) The universality, equivalence and inclusion problem for ϵ -NWA are EXPTIME-complete.

(c) Deciding, given an ϵ -NWA A and a DNWA B , whether $L(A) \subseteq L(B)$ is PTIME-complete with respect to logspace reductions.

Proof. These complexity properties mostly follow from the corresponding results for NWA without ϵ -transitions, proven in [3] (Proposition 6.1 and Theorem 6.2).

For (a), ϵ -NWA may also be interpreted as pushdown automata, whose membership and emptiness problem are decidable in polynomial time.

Lower bounds for (b) directly follow from the corresponding lower bounds for NWA without ϵ -transitions. Corresponding upper bounds can be proven using Lemmas A.4 and A.5 as well as the complement construction for DNWA ([3], Theorem 3.5) and set-theoretic formulations for universality (is $L(A) = \emptyset$?), equivalence (is $(\overline{L(A_1)} \cap L(A_2)) \cup (L(A_1) \cap \overline{L(A_2)}) = \emptyset$?) and inclusion (is $L(A_1) \cap \overline{L(A_2)} = \emptyset$?).

The upper bound in (c) also follows from the fact that DNWA can be efficiently complemented and the fact that $L(A) \subseteq L(B)$ holds if and only if

$L(A) \cap \overline{L(B)} = \emptyset$. By Lemma A.5 and part (a) of this theorem, this can be checked in polynomial time.

The lower bound in (c) is proven by reduction from the emptiness problem for DNWA (cf. Theorem A.2). Let B be a DNWA to be checked for emptiness. We can construct in logarithmic space an ϵ -NWA A deciding $\text{NW}(\Sigma)$, and (since B is deterministic) a DNWA B' deciding the complement of $L(B)$. It then holds that $L(A) \subseteq L(B')$ if and only if $L(B') = \text{NW}(\Sigma)$, which is the case if and only if $L(B)$ is empty. \square

Proofs for Section 3

We begin by defining the formal semantics of NWT.

Definition A.7. Let $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ be an NWT. A *configuration* $\kappa = (q, \alpha)$ consists of a linear state $q \in Q$ and a stack $\alpha \in P^*$ of hierarchical states.

For any string $w \in \text{NW}(\Sigma)$, an ϵ -*extension* of w is a string \hat{w} obtained by inserting symbols $\langle \epsilon \rangle$, $\langle / \epsilon \rangle$ and ϵ into w such that the maximal subsequence of \hat{w} consisting only of symbols from $\hat{\Sigma} \cup \{\langle \epsilon \rangle, \langle / \epsilon \rangle\}$ is a well-nested word over $\Sigma \cup \{\epsilon\}$.

A *run* of T on an ϵ -extension $\hat{w} = \hat{w}_1 \dots \hat{w}_n$ of a string $w \in \text{NW}(\Sigma)$ starting at configuration (r_0, α_0) is a finite sequence $\rho = (r_0, \alpha_0)(r_1, \alpha_1) \dots (r_n, \alpha_n)$ such that for each $i \in [n]$, one of the following holds:

- $\hat{w}_i = \langle a \rangle$ for some $a \in \Sigma \cup \{\epsilon\}$, $(r_i, p_i, u_i) \in \delta(r_{i-1}, \langle a \rangle)$ and $\alpha_i = \alpha_{i-1}p$,
- $\hat{w}_i = \epsilon$, $(r_i, u_i) \in \delta(r_{i-1}, \epsilon)$ and $\alpha_i = \alpha_{i-1}$, or
- $\hat{w}_i = \langle / a \rangle$ for some $a \in \Sigma \cup \{\epsilon\}$, $(r_i, u_i) \in \delta(r_{i-1}, p, \langle / a \rangle)$ and $\alpha_i p = \alpha_{i-1}$.

The run ρ is *accepting* if $r_0 = q_0$, $r_n \in F$, and $\alpha_0 = \alpha_n = \epsilon$; in this case, the string $u_1 u_2 \dots u_n$ is considered the *output* of T on w according to ρ .³

Note that the semantics for NWT directly carry over to ϵ -free NWT, with the only ϵ -extension of any $w \in \mathcal{D}(T)$ on which there are accepting runs of T being w itself.

Lemma 5 (restated). *Each NWT $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ can be transformed in polynomial time into an NWT $T' = (Q', P', P'_\epsilon, \Sigma, \delta', q_0, F)$ with $T(w) = T'(w)$ for each $w \in \text{NW}(\Sigma)$, such that for any transition in δ' with output u , it holds that $|u| \leq 1$.*

We say that a NWT of this shape is in normal form.

Proof. An arbitrary NWT T is transformed into an NWT T' in normal form by successively replacing each transition that is not of the required form by a sequence of new states and transitions. We only describe this procedure for an opening transition; closing and internal transitions can be handled in a similar manner.

Assume for some $q \in Q$ and $a \in \Sigma$ that $(q', p', v) \in \delta(q, \langle a \rangle)$ with $v = v_1 \dots v_n \in \hat{\Sigma}^*$ for some $n > 1$. We add new states q_1, \dots, q_{n-1} to Q and $p_{p', a}$ to P_ϵ . Let $k \leq n$ be the position of the last unmatched opening tag in v , i.e. $v_1 \dots v_{k-1} \in \hat{\Sigma}^*$, $v_k \in \langle \Sigma \rangle$ and $v_{k+1} \dots v_n \in \text{NW}(\Sigma)$. We add a transition

³Note that the ϵ -extension \hat{w} on which the output $u_1 u_2 \dots u_n$ is produced is already implicit in the run ρ , so we do not specify it explicitly.

$(q_1, p_{p',a}, v_1)$ to $\delta(q, \langle \epsilon \rangle)$ ⁴, a transition (q_{k+1}, p', v_k) to $\delta(q_k, \langle a \rangle)$ and, for each $i \in [n-1]$ with $i \neq k$, a transition $(q_{i+1}, p_{p',a}, v_i)$ to $\delta(q_i, \langle \epsilon \rangle)$ if $v_i \in \langle \Sigma \rangle$, or (q_{i+1}, v_i) to $\delta(q_i, p_{p',a}, \langle \epsilon \rangle)$ if $v_i \in \langle \Sigma \rangle$, identifying q_n with q' . Finally, we remove the original transition. This takes care of (reading or ϵ -)transitions producing more than one output symbol.

The resulting NWT T' is obviously in normal form. Its equivalence to T is relatively simple (if tedious) to prove by an induction over the structure of input strings, with the main argument using the fact that T fulfils the ϵ -consistency and well-formedness condition. Notably, these conditions also justify the above simplification that newly added ϵ transitions obtained from a transition reading a and pushing the hierarchical state p' use solely the new hierarchical ϵ -state $p_{p',a}$. \square

Lemma 6 (restated). *Any context-free game $G = (\Sigma, \Gamma, R, T)$ with NWT R can be transformed in polynomial time into a game $G' = (\Sigma', \Gamma, R', T')$ such that R' is non-deleting and it holds that $JWin(G') \cap NW(\Sigma) = JWin(G)$.*

Proof. (Sketch) The idea behind this proof is modifying R into R' in such a way that, whenever R would delete some tag $\langle a \rangle$ (or $\langle \epsilon/a \rangle$), R' instead replaces tag by a special “strike-out” version $\langle \# \rangle$ (or $\langle \epsilon/\# \rangle$) with $\# \notin \Sigma$, instead; we therefore set $\Sigma' = \Sigma \uplus \{\# \mid a \in \Sigma\}$. To ensure that iterated transductions respect deleted tags, we add transitions to each state of R' that only replace “strike-out” tags by themselves without changing the state of R' . Finally, we similarly modify the target DNWA for T in order to ignore all tags with labels not in Σ (i.e. only check them for their nesting structure without changing states). Since JULIET may only play Call on symbols from Σ , ROMEO is unable to rewrite “strike-out” symbols, and the DNWA for T' ignores symbols outside of Σ , it is clear that JULIET has a winning strategy on any string $w \in NW(\Sigma)$ in G' if and only if she has a winning strategy on w in G . \square

Proposition 7 (restated). *Let T_1, T_2 be non-deleting NWT. Then, there exists a non-deleting NWT T such that for all $w \in NW(\Sigma)$, it holds that $T(w) = (T_2 \circ T_1)(w) \stackrel{\text{def}}{=} T_2(T_1(w))$. This NWT T can be computed from T_1 and T_2 in polynomial time and is of size $O(|T_1| \cdot |T_2|)$.*

Proof. The basic idea behind this construction is simple: The transducer T simulates T_1 on its input and directly feeds the output of T_1 into T_2 . This is done by a sort of product construction which is, for the most part, quite straightforward. However, there are some subtleties that need to be addressed, which stem from the fact that both T_1 and T_2 are NWT, so the various possibilities how one of the two transducers might perform a step while the other is idle have to be dealt with. We note that this proof could be extended to general (not necessarily non-deleting) NWT, but since the proof details are quite technical already, we restrict our attention to non-deleting NWT here.

Let $T_1 = (Q^1, P^1, P_\epsilon^1, \Sigma, \delta^1, q_0^1, F^1)$ and $T_2 = (Q^2, P^2, P_\epsilon^2, \Sigma, \delta^2, q_0^2, F^2)$ be two NWT in normal form. We construct from T_1 and T_2 the NWT $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ as follows:

- $Q = Q^1 \times Q^2$,

⁴Note that $v_1 \in \langle \Sigma \rangle$ due to well-formedness.

- $P = (P^1 \cup \{\perp\}) \times P^2$ for a special symbol \perp not used in T_1 or T_2 ,
- $P_\epsilon = (P_\epsilon^1 \times P^2) \cup (\{\perp\} \times P_\epsilon^2)$
- $q_0 = (q_0^1, q_0^2)$,
- $F = F^1 \times F^2$, and
- δ is constructed as detailed below.

The construction of δ is quite straightforward in the case where T_1 produces an output on which T_2 is simulated: If $(q'_1, p'_1, \langle b \rangle) \in \delta^1(q_1, \langle a \rangle)$ (or $(q'_1, p'_1, \langle b \rangle) \in \delta^1(q_1, \langle \epsilon \rangle)$, respectively) and $(q'_2, p'_2, \langle c \rangle) \in \delta^2(q_2, \langle b \rangle)$, then $((q'_1, q'_2), (p'_1, p'_2), \langle c \rangle) \in \delta((q_1, q_2), \langle a \rangle)$ (or $((q'_1, q'_2), (p'_1, p'_2), \langle c \rangle) \in \delta((q_1, q_2), \langle \epsilon \rangle)$, respectively), and analogously for the corresponding closing (ϵ -) transitions.

Internal ϵ -transitions for T_1 and T_2 are also easily handled, as for these transitions, neither transducer consumes an input or produces an output. If $(q'_1, \epsilon) \in \delta^1(q_1, \epsilon)$, then $((q'_1, q_2), \epsilon) \in \delta((q_1, q_2), \epsilon)$ for each $q_2 \in Q^2$, and if $(q'_2, \epsilon) \in \delta^2(q_2, \epsilon)$, then $((q_1, q'_2), \epsilon) \in \delta((q_1, q_2), \epsilon)$ for each $q_1 \in Q^1$.

The only case requiring special attention is the one where only one of the two transducers modifies the input. This happens when T_2 produces an output symbol by an ϵ -transition, without T_1 consuming an input symbol. In this case, T_2 produces a hierarchical state while T_1 doesn't; to this end, some states in the set P_ϵ contain a component \perp to indicate a “null transition” for T_1 .

More formally, if for some $q_2, q'_2 \in Q^2$, $p'_2 \in P^2$ and $a \in \Sigma$, $(q'_2, p'_2, \langle a \rangle) \in \delta^2(q_2, \langle \epsilon \rangle)$, then $((q_1, q'_2), (\perp^1, p'_2), \langle a \rangle) \in \delta((q_1, q_2), \langle \epsilon \rangle)$ for all $q_1 \in Q^1$ (and analogously for closing transitions).

It follows directly from the construction that T is ϵ -consistent; the well-formedness condition for T follows from well-formedness of T_1 and T_2 by a simple but lengthy case distinction over all the sorts of transitions introduced here. Furthermore, since T is obviously in normal form, it automatically fulfils synchronisation. It remains to be proven that $T(w) = T_2(T_1(w))$ indeed holds for all $w \in \text{NW}(\Sigma)$.

To this end, let $w_2 \in T_2(T_1(w))$; let further ρ_1 be an accepting run of T_1 with output w_1 on an ϵ -extension \hat{w} of w , and let ρ_2 be an accepting run of T_2 with output w_2 on an ϵ -extension \hat{w}_1 of w_1 . We construct an ϵ -extension \hat{w}' of w and an accepting run ρ of T on \hat{w}' with output w_2 .

We denote all positions of \hat{w} in which T_1 outputs some symbol as *1-producing*. Note that there is a bijective correspondence between positions of w_1 and 1-producing positions of \hat{w} and that all $\langle \epsilon \rangle$ - and $\langle / \epsilon \rangle$ -positions of \hat{w} are 1-producing. Next, we examine all $\langle \epsilon \rangle$ - and $\langle / \epsilon \rangle$ -positions of \hat{w}_1 ; these, we call *2-producing*. Due to the well-formedness and ϵ -consistency restrictions on T_1 and T_2 , it is possible to insert all $\langle \epsilon \rangle$ -, $\langle / \epsilon \rangle$ - and ϵ -positions of \hat{w}_1 into \hat{w} in such a way that we obtain an ϵ -extension \hat{w}' of w that has both \hat{w} and \hat{w}_1 as subsequences.

It is now easy to see that an accepting run ρ of T on \hat{w}' with output w_2 can be obtained by combining the transitions used in ρ_1 and ρ_2 – positions in \hat{w}' that are 1-producing correspond to “standard” transitions of T , ϵ -positions correspond to internal ϵ -transitions, and 2-producing positions correspond to ϵ -transitions with a \perp component in their hierarchical state. This shows that $T_2(T_1(w)) \subseteq T(w)$.

For the other direction, let ρ be a run of T on an ϵ -extension \hat{w}' of w with output w_2 . We label the positions of \hat{w}' according to the transitions taken

by T in ρ – positions where “standard” transitions are used are labelled as 1-producing, ϵ -positions are labelled as internal, and positions with transitions whose hierarchical stack contains a \perp component are labelled as 2-producing. Similar to the previous part of the proof, we can then use these labels to separate ρ into an accepting run ρ_1 of T_1 on an ϵ -extension \hat{w} of w with output w_1 and an accepting run ρ_2 of T_2 on an ϵ -extension \hat{w}_1 of w_1 with output w_2 , thus proving that $w_2 \in T_2(T_1(w))$ holds. \square

Corollary 8 (restated). *Let T be a non-deleting NWT and A a NWA over alphabet Σ . Then, there exists a non-deleting NWT T' of size $\mathcal{O}(|T| \cdot |A|)$ such that $\mathcal{D}(T') = \mathcal{D}(T) \cap L(A)$ and $T'(w) = T(w)$ for each $w \in \mathcal{D}(T) \cap L(A)$.*

Proof. Let T_A be a NWT with $\mathcal{D}(T_A) = L(A)$ and $T_A(w) = \{w\}$ for each $w \in \text{NW}(\Sigma)$, i.e. T_A accepts exactly the strings in $L(A)$ and simply outputs its input string. Such a NWT is easy to construct with a size in $\mathcal{O}(|A|)$.

Then, we set T' to be the NWT for $T \circ T_A$ as constructed in Proposition 7. For each $w \in L(A) \cap \mathcal{D}(T)$, it holds that $T'(w) = T(T_A(w)) = T(w)$, for each $w \notin L(A)$, $T'(w) = T(\emptyset) = \emptyset$, and for each $w \in L(A) \setminus \mathcal{D}(T)$, it holds that $T'(w) = T(w) = \emptyset$. This proves that $\mathcal{D}(T') = \mathcal{D}(T) \cap L(A)$ and $T'(w) = T(w)$ for each $w \in \mathcal{D}(T) \cap L(A)$. The desired bound on the size of T' follows directly from Proposition 7. \square

Lemma 9 (restated). *Let T be a non-deleting NWT with $\mathcal{D}(T) \subseteq \text{NW}(\Sigma)$. Then $\mathcal{R}(T)$ is a regular language of nested words.*

Proof. Let $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ be a non-deleting NWT in normal form. The basic idea behind constructing the ϵ -NWA A for $\mathcal{R}(T)$ is taking the input string for A and verifying it against the output component of T . This way, it is easy to ensure that every string in $\mathcal{R}(T)$ is accepted by A ; some care has to be taken, however, to make certain that for every string w' accepted by A , there is a string $w \in \text{NW}(\Sigma)$ such that $w' \in T(w)$.

Since T is in normal form, all opening transitions in δ are of the form $(q, \langle a \rangle, q', p, \langle b \rangle)$ and all closing transitions of the form $(q, p, \langle /a \rangle, q', \langle /b \rangle)$ (for $a \in \Sigma \cup \{\epsilon\}$ and $b \in \Sigma$) while internal transitions are of the form $(q, \epsilon, q', \epsilon)$.

The linear state set of A is Q and its set of hierarchical states is $P \times (\Sigma \cup \{\epsilon\})$; its starting and accepting states are those of T . Reading transitions in A are constructed from those of T by taking, for each opening transition $(q, \langle a \rangle, q', p, \langle b \rangle) \in \delta$, a transition $(q, \langle b \rangle, q', (p, a))$, and for each closing transition $(q, p, \langle /a \rangle, q', \langle /b \rangle) \in \delta$, a transition $(q, (p, a), \langle /b \rangle, q')$. Each internal transition $(q, \epsilon, q', \epsilon) \in \delta$ translates to an ϵ -transition (q, ϵ, q') in A .

To prove correctness of this construction, we need to show that for any string $w' \in \text{NW}(\Sigma)$ it holds that $w' \in L(A)$ if and only if $w' \in \mathcal{R}(T)$.

For the “if” direction, if $w' \in \mathcal{R}(T)$, then there is some $w \in \text{NW}(\Sigma)$ such that T has an accepting run ρ on some ϵ -extension of w with output w' . Translating the transitions of T taken in ρ into transitions of A as per the above construction naturally yields an accepting run of A on w' .

For the “only if” direction, assume that $w' \in L(A)$ for some $w' \in \text{NW}(\Sigma)$. Then, there is an accepting run ρ of A on w' . We can construct from ρ an ϵ -extension \hat{w} of a string $w \in \text{NW}(\Sigma)$ by extracting the sequence of hierarchical components of transitions in ρ – every opening (closing) transition in ρ with hierarchical component (p, a) with some $p \in P$, $a \in \Sigma \cup \{\epsilon\}$ corresponds to

a symbol $\langle a \rangle$ ($\langle /a \rangle$) in \hat{w} , and every ϵ -transition taken in ρ corresponds to a symbol ϵ in \hat{w} . It is clear to see that indeed \hat{w} is an ϵ -extension of some string $w \in \text{NW}(\Sigma)$ and that there is an accepting run of T on \hat{w} that outputs w' , which yields $w' \in \mathcal{R}(T)$ as was to be proven. \square

Corollary 10 (restated). *Regular nested word languages are closed under transduction by non-deleting NWT, i.e. if $L \subseteq \text{NW}(\Sigma)$ is regular and T a NWT, then $T(L)$ is regular.*

Proof. Let A be a NWA for L and T a NWT. From A , we can construct by Corollary 8 a NWT T_A with $\mathcal{D}(T_A) = L \cap \mathcal{D}(T)$ and $T_A(w) = T(w)$ for each $w \in \mathcal{D}(T_A)$, which implies that $\mathcal{R}(T_A) = T(L)$. By Lemma 9, this implies that $T(L)$ is regular as well. \square

Theorem 11 (restated). *The membership problem for non-deleting NWT (Given a non-deleting NWT T and strings $w, u \in \text{NW}(\Sigma)$, is $u \in T(w)$?) is in PTIME.*

Proof. From w , we can easily compute a NWA A of size $\mathcal{O}(w)$ with $L(A) = \{w\}$. By Corollary 8, we can compute from A and T in polynomial time a non-deleting NWT T_w with $\mathcal{D}(T_w) = \mathcal{D}(T) \cap \{w\}$ and $\mathcal{R}(T_w) = T(w)$. By Corollary 10, $T(w)$ is regular, and by Lemma 9 an ϵ -NWA A' for $T(w)$ can be computed in polynomial time. This ϵ -NWA is of polynomial size, and checking u for membership in $L(A)$ is possible in polynomial time by Theorem A.6 (a). \square

Theorem 12 (restated). *The nonemptiness problem for non-deleting NWT (Given a non-deleting NWT T , is there a string $w \in \text{NW}(\Sigma)$ with $T(w) \neq \emptyset$?) is PTIME-complete with regard to logspace reductions.*

Proof. By definition, it holds that there is some $w \in \text{NW}(\Sigma)$ with $T(w) \neq \emptyset$ if and only if $\mathcal{D}(T) \cap \text{NW}(\Sigma) \neq \emptyset$. By Corollary 8, we can compute from T a NWT T' whose domain is exactly $\mathcal{D}(T) \cap \text{NW}(\Sigma)$. Clearly, it holds that $\mathcal{D}(T') \neq \emptyset$ if and only if $\mathcal{R}(T') \neq \emptyset$, so by Lemma 9 we can extract from T' an ϵ -NWA A of polynomial size with $L(A) = \mathcal{R}(T')$. All of these transformations, as well as testing whether $L(A) \neq \emptyset$, are feasible in polynomial time. This proves the upper bound.

The lower bound follows by reduction from the emptiness problem for DNWA (cf. Theorem A.2). Let $A = (Q, P, \Sigma, \delta, q_0, F)$ be a DNWA to be tested for non-emptiness. We construct from A a NWT T with $T(\epsilon) = L(A)$ and $T(w) = \emptyset$ for all $w \neq \epsilon$. The idea behind the reduction is replacing, for each $a \in \Sigma$, each $\langle a \rangle$ (resp. $\langle /a \rangle$) transition of A by an $\langle \epsilon \rangle$ (resp. $\langle / \epsilon \rangle$) transition in T that writes $\langle a \rangle$ (resp. $\langle /a \rangle$) as an output. Formally, we set $T = (Q, P \times \Sigma, P \times \Sigma, \Sigma, \delta', q_0, F)$, where δ' contains a transition $(q, \langle \epsilon \rangle, q', (p, a), \langle a \rangle)$ (resp. $(q, (p, a), \langle / \epsilon \rangle, q', \langle /a \rangle)$) if and only if δ contains a transition $(q, \langle a \rangle, q', p)$ (resp. $(q, p, \langle /a \rangle, q')$). This construction is feasible using logarithmic space, and it holds that $T(\epsilon) = L(A)$ and $T(w) = \emptyset$ for all $w \neq \epsilon$, so $\mathcal{D}(T) \neq \emptyset$ if and only if $L(A) \neq \emptyset$. \square

Theorem 13 (restated). *The type checking problem for non-deleting NWT (Given a non-deleting NWT T and NWA A_1, A_2 , is $T(L(A_1)) \subseteq L(A_2)$?) is*

- (a) EXPTIME-complete in general, and
- (b) PTIME-complete (w.r.t. logspace reductions) if A_2 is a DNWA.

Proof. The lower bound in (a) follows directly by a reduction from the inclusion problem for NWA, which is known to be EXPTIME-complete [3]. Let B_1 and B_2 be NWA to be checked for inclusion of $L(B_1)$ in $L(B_2)$. We can construct in polynomial time a NWT T that simply reproduces each input symbol in the output and accepts any input (i.e. $T(w) = \{w\}$ for any $w \in \hat{\Sigma}^*$). Then, clearly, $L(B_1) \subseteq L(B_2)$ if and only if $T(L(B_1)) \subseteq L(B_2)$, as $T(L(B_1)) = L(B_1)$.

The lower bound in (b) follows from PTIME-hardness of the emptiness problem for DNWA by a similar reduction to that used in the proof of Theorem A.6 (c). Let A be a DNWA to be tested for emptiness. We can construct in logarithmic space an ϵ -NWA A_1 deciding $\text{NW}(\Sigma)$, and (since A is deterministic) a DNWA A_2 deciding the complement of $L(A)$. For T , we construct a NWT reproducing its input, i.e. with $\mathcal{D}(T) = \text{NW}(\Sigma)$ and $T(w) = \{w\}$ for each $w \in \text{NW}(\Sigma)$. It then holds that $T(L(A_1)) \subseteq L(A_2)$ if and only if $L(A_2) = \text{NW}(\Sigma)$, which is the case if and only if $L(A)$ is empty.

The upper bounds in both (a) and (b) can easily be proven using prior results from this section. By Corollary 8 and Lemma 9, we can construct in polynomial time an ϵ -NWA deciding $T(L(A_1))$; checking this ϵ -NWA for inclusion in $L(A_2)$ is generally possible in exponential time, and in polynomial time if A_2 is a DNWA by Theorem A.6 (b) and (c). \square

Proofs for Section 4

Theorem 14 (restated). *For the class of games with NWT and Call depth $k \geq 2$, JWIN is not recursively enumerable.*

Proof. The proof is by reduction from the complement of the halting problem for Turing machines with an empty input. From a given Turing machine M with working alphabet Σ and state set Q , we construct a game G and string w such that JULIET has a winning strategy on w in G if and only if M does not halt on an empty input. The game G uses two function symbols, s and t , which are not in Σ in order to avoid plays according to G interfering with the workings of M .

The idea behind the reduction is rather simple. The game begins on the input string $\langle s \rangle \langle /s \rangle$, where JULIET is supposed to Call $\langle /s \rangle$ as her first move. For his reply, ROMEO then picks a number r such that M halts on the empty input after exactly r steps (if such a number exists) and returns $\langle t \rangle^r \hat{v}_0 \langle /t \rangle^r$, with \hat{v}_0 being a string representing the initial configuration of M on the empty input. JULIET should then call each $\langle /t \rangle$ in left-to-right order, making ROMEO simulate a step of M on the current position of M encoded in the current string. After JULIET has called all r closing $\langle /t \rangle$ tags and thus rewritten \hat{v}_0 into a string \hat{v}_r representing the configuration of M after r computation steps, JULIET wins the game if \hat{v}_r does not represent a halting configuration.

We can represent configurations of M as flat strings over the alphabet $\Sigma \times (Q \cup \{-\})$ in the standard fashion – a string $(x_1, -) \cdots (x_{k-i}, -)(x_k, q)(x_{k+1}, -) \cdots (x_m, -)$ denotes that the content of M 's working tape is $x_1 \cdots x_m$, with the head of M being on the tape's k -th cell and M being in state q . Each such flat string v can then be represented as a nested string \hat{v} using the standard nested word encoding defined in the preliminaries. Without loss of generality, we assume that M always moves its head to the left-most used tape cell before halting, i.e. the flat string encoding of a halting configuration is of the form $(\Sigma \times \{h\})(\Sigma \times \{-\})^*$ for the halting state h of M .

It is easy to construct a polynomial-sized NWT rewriting $\langle s \rangle \langle /s \rangle$ into a string of the form $\langle t \rangle^r \hat{v}_0 \langle /t \rangle^r$ (for arbitrary $r \in \mathbb{N}$ and a nested string encoding \hat{v}_0 of M 's initial configuration) and every string $\langle t \rangle \hat{v}_i \langle /t \rangle$ where \hat{v}_i is the nested word encoding of some configuration of M into \hat{v}_{i+1} , where \hat{v}_{i+1} encodes the successor configuration of the one encoded by \hat{v}_i ; in fact, the latter rewriting is even functional (if nondeterministic for requiring a look-ahead when M moves its head to the left). We can also easily construct a DNWA accepting all nested-word encodings of non-halting configurations of M .

It only remains to be explained how we ensure that the game is played in the fashion sketched above, i.e. that JULIET plays Call first on $\langle /s \rangle$ and then on each $\langle /t \rangle$ in left-to-right order. To keep JULIET from leaving $\langle /s \rangle$ or some $\langle /t \rangle$ uncalled, we simply set up the target DNWA such that it doesn't accept any nested strings containing tags with labels s or t . Finally, to safeguard against JULIET skipping some $\langle /t \rangle$ before calling the next, we modify the replacement transducer R in such a way that it rejects any input string containing two or more nested t tags, i.e. strings of the form $\langle t \rangle \langle t \rangle v \langle /t \rangle \langle /t \rangle$ with $v \in \text{NW}(\Sigma)$.

In the game G thus constructed, JULIET clearly has a winning strategy on $w = \langle s \rangle \langle /s \rangle$ if and only if there is no r such that M reaches a halting configuration within r steps, i.e. if M doesn't halt. \square

We now prove the lower and upper bounds for Theorem 15. Since context-free games with general and with non-deleting NWT are polynomially equivalent by Lemma 6, we prove the lower bound for general NWT and the lower bound for non-deleting NWT.

Proposition A.8. For the class of replay-free games with NWT, JWIN is 2-EXPTIME-hard.

Proof. The proof is by reduction from the complement of the 2-PLAYER EXPONENTIAL CORRIDOR TILING problem. In this problem, we are given a tiling instance consisting of a tile set U , vertical and horizontal constraints V, H , initial and final tile u_i, u_f and unary number n and consider the following game: Player 1 and Player 2 place tiles from U in an alternating fashion (starting with u_i for Player 1); Player 1 wins the game if Player 2 places a tile violating some vertical or horizontal constraint, or some player places a tile completing a valid tiling of width $2^n - 1$, while Player 2 wins if Player 1 places an invalid tile or if the game does not terminate with a valid tiling. 2-PLAYER EXPONENTIAL CORRIDOR TILING asks whether, given a tiling instance, Player 1 has a winning strategy in this game. This problem (and therefore also its complement problem of determining whether Player 2 has a winning strategy) is complete for $\text{AEXPSPACE} = 2\text{-EXPTIME}$ [6].

Given a tiling instance $\mathcal{I} = (U, V, H, u_i, u_f, n)$, we construct a game $G = (\Sigma, \Gamma, R, T)$ and string w such that JULIET has a winning strategy in G on w if and only if Player 2 has a winning strategy in the exponential-width tiling game on instance \mathcal{I} .

The input string is $w = (\langle d \rangle \langle e \rangle)^n \langle s \rangle \langle /s \rangle (\langle /e \rangle \langle /d \rangle)^n$. The basic idea behind the game G is that JULIET is first supposed to play Call on $\langle /s \rangle$, to which ROMEO should give a witness for the existence of a winning strategy for Player 1 in the tiling game. JULIET should then use calls to the rest of the input string to uncover a flaw in ROMEO's witness and thus have a winning strategy in G on w if and only if ROMEO cannot prove the existence of a winning strategy for Player 1.

More concretely, ROMEO should give as his witness (a slightly modified linearisation of) a *strategy tree* for Player 1, in which nodes are labelled with tiles from two disjoint copies U^1, U^2 of U , representing moves by Player 1 and Player 2. Each node labelled with an element of U^2 has a single child (corresponding to a move of Player 1), each node labelled with an element of U^1 has $|U|$ children (corresponding to the possible moves of Player 2) and the sequence of labels on each path from the root to a leaf (called a *tiling candidate*) either makes up a valid tiling or contains an invalid move by Player 2. Replacement strings for ROMEO on a Call to $\langle /s \rangle$ will be somewhat extended linearisations of strategy trees, with extensions that we will discuss later on.

Proof trees given by ROMEO should be restricted (by construction of the replacement transducer R) to only represent tiling candidates that start with the initial tile u_i^1 and contain no horizontal errors. JULIET's task in trying to invalidate ROMEO's proof tree therefore consists of uncovering vertical errors and incorrect final tiles (as well as encoding errors within the aforementioned extensions, to be discussed later). To this end, JULIET uses Call moves on $\langle /e \rangle$ and $\langle /d \rangle$ tags to select parts of the proof tree given by ROMEO that correspond to a fixed column in each tiling candidate represented in the proof tree; if this column contains an error in at least one of these tiling candidates (which should be detected by the target language DNWA), JULIET wins the game.

We now describe how strategy trees should be encoded. Throughout the rest of the proof, we will identify nested words and their forest representations to simplify presentation.

The most intuitive approach to generate strategy trees in which all tiling candidates are horizontally correct would be to fix the replacement transducer in such a way that each node labelled with some $u^1 \in U^1$ has only children labelled by some $v^2 \in U^2$ with $(u, v) \in H$; additionally, to make certain that a strategy tree correctly represents all possible counter-strategies for Player 2, we would have to require that each node corresponding to a move by Player 2 has as children nodes with *all* possible labels v^2 such that $(u, v) \in H$. This intuitive approach leads to two minor problems, though:

- For some tiles u , there may be no tiles v with $(u, v) \in H$. If this happens, we fix the replacement transducer such that it may only follow u up with a special “pseudo-tile” $u_{\text{err}} \notin U$.
- If a tile u is placed at the end of a line, the next tile v does *not* have to fulfil $(u, v) \in H$. In order to account for this fact and still have the replacement transducer produce horizontally correct tiling candidates, we introduce a special *line divider* symbol $\# \notin U$ which may be placed by the replacement transducer at any time it could normally place a node for some player, regardless of its parent tile, and which may be followed up by either *any* tile $u^1 \in U^1$ as a move for Player 1, or by *all* tiles $u^2 \in U^2$ as possible moves for Player 2.

In a correct encoding of a strategy tree, the line divider symbol $\#$ should occur as a node label if and only if the corresponding node’s depth is a multiple of 2^n . However, the above construction of replacement transducer allows ROMEO to construct incorrect encodings of strategy trees. Dealing with attempts by ROMEO to cheat in this manner, as well as with vertical errors, is what we discuss next.

We interpret a tree given by ROMEO as a tree whose paths from root to leaf encode tiling candidates with 2^n columns and an arbitrary number of lines (i.e. root-to-leaf paths should always have as their length a multiple of 2^n), with the last column in a correctly encoded tiling candidate consisting only of line divider symbols, and no line divider symbol occurring in any other column.

As mentioned above, to check for vertical errors and incorrect encodings, JULIET selects some column number $\ell \in [2^n]$ and removes all nodes not corresponding to some tile in that column, i.e. reduces the original tree t to one containing only the nodes at depths $k \cdot (2^n) + \ell$ for all $k \geq 0$. How this is done exactly will be explained later on. Due to the alternation between Players 1 and 2, if the column chosen by JULIET is not the last column, each node with a label from U^1 should only have children with labels from U^2 in this reduced tree, or vice versa.

At first glance, it seems clear that, with the considerations on encoding made above, JULIET can already uncover attempts by ROMEO to cheat or give a non-winning strategy, namely

- by picking a column where $\ell \neq 2^n$ and in some path of the resulting tree, there are two directly subsequent labels from U^1 or from U^2 (ROMEO tried to cheat by disrespecting the correct player sequence after a $\#$), or

- by picking a column $\ell \neq 2^n$ in which some path contains a $\#$, (ROMEO tried to cheat by placing an incorrect $\#$), or
- by picking a column where $\ell \neq 2^n$ and in some path of the resulting tree, there is a tile $u^2 \in U^2$ followed either by u_{err} or some $v^1 \in U^1$ with $(u, v) \notin V$ (the strategy given by ROMEO is non-winning due to a horizontal or vertical error), or
- by picking $\ell = 2^n - 1$ when for some path in the resulting tree, its leaf is not u_f (the strategy given by ROMEO is non-winning due to a wrong final tile), or
- by picking $\ell = 2^n$ if the last column contains some symbol that is not $\#$ in some path (ROMEO tried to cheat by not placing a correct $\#$).

All of these conditions can easily be checked by a polynomial-size DNWA.

However, while these conditions are indeed sufficient for checking whether ROMEO has given an incorrect or non-winning strategy tree, our construction so far is still too restrictive in that it sometimes fails to recognise a correct winning strategy tree. Consider, for instance, the case where all paths of the strategy tree either contain a vertical error for Player 2 or end with a valid tiling. According to our construction so far, JULIET could still win on a strategy tree of this shape, as our construction requires ROMEO to continue giving horizontally correct tiles even after a vertical error has occurred, and these “irrelevant” tiles might not always make it possible to end a line with u_f , thus allowing JULIET to find some path in the strategy tree corresponding to column $2^n - 1$ that doesn’t end in u_f .

To address this problem, we further modify the replacement transducer R in such a way that, whenever it is supposed to output a node with label $u^2 \in U^2$ corresponding to a move by Player 2, it may instead nondeterministically choose to output a *marked* version \hat{u} of u^2 instead. This is supposed to indicate that choosing u^2 leads to a vertical error for Player 2, so below the node labelled \hat{u} , the replacement transducer may produce a sequence of arbitrary tiles from U^1 for Player 1 and from U^2 for Player 2 (as vertically correct “pseudo-tiles”), terminating with u_f at an odd depth followed by $\#$. If a node labelled u_{err} ever occurs instead of a move of Player 2 due to a horizontal error, it is followed up by a similar path.

In this way, we make it ROMEO’s responsibility to flag vertical errors for Player 2. This would allow ROMEO to cheat by claiming a vertical error when, in fact, there is none, but such attempts at cheating can be penalised by once again adapting the target language in such a way that JULIET wins if she selects a column in which ROMEO has falsely flagged a vertical error (i.e. the selected column number ℓ is not 2^n and some path in the corresponding tree contains a marked tile \hat{v} that is not preceded by some $u^1 \in U^1$ with $(u, v) \notin V$).

Next, we examine how JULIET selects a column in the strategy tree given by ROMEO. Recall that the input string is $w = (\langle d \rangle \langle e \rangle)^n \langle s \rangle \langle /s \rangle (\langle /e \rangle \langle /d \rangle)^n$, and that the first move by JULIET is supposed to be a Call on $\langle /s \rangle$ to replace $\langle s \rangle \langle /s \rangle$ by an encoding of a strategy tree, in which each path is to be interpreted as a tiling candidate with $2^n - 1$ columns and an additional column made up of separator symbols $\#$. JULIET now selects one of these 2^n columns by incrementally causing ROMEO to delete either all odd-numbered columns or all even-numbered

columns (i.e. all nodes at odd or even depths in the strategy tree) until only a single column is left. This is the purpose of the nodes labelled d and e .

By a Call to some node labelled e , JULIET causes ROMEO to change its label to o (without changing the tree nested below it). A Call to a d -node with e -labelled child causes ROMEO to delete both of these nodes as well as all nodes at even depths in the strategy tree nested below them; similarly, a Call to a d -node with o -child deletes these nodes and all odd-depth nodes in the strategy tree.

To keep JULIET from cheating in this selection process, we have to make certain that she calls *all* d -nodes in left-to-right order. We can easily keep JULIET from leaving uncalled d -nodes by fixing the target language to not contain any strings including the label d ; to make certain that she does not skip any d -nodes, the replacement transducer rejects any strings containing a d -node as child of an e - or o -node.

To later check for the right kinds of vertical errors or inconsistencies, we have to keep track of the column selected by JULIET, i.e. the sequence of Call moves deleting odd or even columns. The following cases have to be distinguished:

- If JULIET deletes odd columns on all n calls, she selects column 2^n ;
- if JULIET deletes even columns as her first call and then only odd columns on the following $n - 1$ calls, she selects column $2^n - 1$;
- otherwise, JULIET selects a column with index $\ell \leq 2^n - 2$.

To store this information, we use an additional node, initially labelled 0, which ROMEO returns as the root of his chosen strategy tree after JULIET's Call on $\langle /s \rangle$. This node is then updated throughout the column-deletion process as follows:

- A label 0 signifies that no deletions have been made so far; on a Call to d which deletes odd columns, 0 is rewritten to l , on a Call deleting even columns, it becomes x
- A label l ("last") signifies that the column JULIET selects may possibly be the 2^n -th line divider column. As long as only odd columns are deleted, the label l remains, if even columns are deleted, l is rewritten to r .
- A label x ("neXt-to-last") signifies that the column JULIET selects may possibly be the (final) column with number $2^n - 1$. As long as only even columns are deleted, the label x remains, if odd columns are deleted, x is rewritten to r .
- A label r ("standaRd") signifies that the column ROMEO selects will definitely not be one of the last two columns, i.e. have a number at most $2^n - 2$. The label r is not rewritten by any deletion.

To summarise, the game $G = (\Sigma, \Gamma, R, T)$ constructed from the tiling instance (U, V, H, u_i, u_f, n) is as follows.

The alphabet of G is

$$\Sigma = U \cup \hat{U} \cup U^1 \cup U^2 \cup \{u_{\text{err}}, \#, s, d, e, o, 0, l, x, r\},$$

(where $\hat{U} = \{\hat{u} \mid u \in U\}$, $U^1 = \{u^1 \mid u \in U\}$ and $U^2 = \{u^2 \mid u \in U\}$) with function symbols $\Gamma = \{s, d, e\}$.

The replacement transducer R behaves as follows:

- R rewrites $\langle s \rangle \langle /s \rangle$ into a string of the form $\langle 0 \rangle w' \langle /0 \rangle$, where w' is the linearisation of a strategy tree as described above, i.e. a tree t with the following properties:
 - t has a root node labelled u_i^1 ;
 - Each node labelled with some $u^1 \in U^1$ that does not have a node with label from $\hat{U} \cup \{u_{\text{err}}\}$ as ancestor has as its children either a single node labelled $\#$, or nodes either labelled v^2 or \hat{v} for each $v \in U$ with $(u, v) \in H$, or a single node labelled u_{err} if no such v exists.
 - Each node labelled with some $u^2 \in U^2$ that does not have a node with label from $\hat{U} \cup \{u_{\text{err}}\}$ as ancestor has a single child labelled either v^1 for some $v \in U$ with $(u, v) \in H$, or u_{err} if no such v exists, or $\#$.
 - Each node that has a node with label from $\hat{U} \cup \{u_{\text{err}}\}$ as its own label or as an ancestor has a single child labelled either by some $u^1 \in U^1$, or by some $u^2 \in U^2$, or by $\#$.
 - Each node labelled with $\#$ has a single child labelled by some $u^1 \in U^1$, or by $u^2 \in U^2$, or no child at all.
- R rewrites strings of the form $\langle e \rangle v \langle /e \rangle$, for arbitrary $v \in \text{NW}(\Sigma)$, into $\langle o \rangle v \langle /o \rangle$.
- R rewrites strings of the form $\langle d \rangle \langle e \rangle v_1 \langle /e \rangle \langle /d \rangle$ into strings v_2 as follows:
 - If $v_1 = \langle 0 \rangle v'_1 \langle /0 \rangle$ (with $v'_1 \in \text{NW}(\Sigma)$), then $v_2 = \langle x \rangle v'_2 \langle /x \rangle$, where v'_2 is derived from v'_1 by deleting all nodes at even depths;
 - If $v_1 = \langle l \rangle v'_1 \langle /l \rangle$ or $v_1 = \langle x \rangle v'_1 \langle /x \rangle$ or $v_1 = \langle r \rangle v'_1 \langle /r \rangle$ (with $v'_1 \in \text{NW}(\Sigma)$), then $v_2 = \langle r \rangle v'_2 \langle /r \rangle$, where v'_2 is derived from v'_1 by deleting all nodes at even depths.
- R rewrites strings of the form $\langle d \rangle \langle o \rangle v_1 \langle /o \rangle \langle /d \rangle$ into strings v_2 as follows:
 - If $v_1 = \langle 0 \rangle v'_1 \langle /0 \rangle$ or $v_1 = \langle l \rangle v'_1 \langle /l \rangle$ (with $v'_1 \in \text{NW}(\Sigma)$), then $v_2 = \langle l \rangle v'_2 \langle /l \rangle$, where v'_2 is derived from v'_1 by deleting all nodes at odd depths;
 - If $v_1 = \langle x \rangle v'_1 \langle /x \rangle$ (with $v'_1 \in \text{NW}(\Sigma)$), then $v_2 = \langle x \rangle v'_2 \langle /x \rangle$, where v'_2 is derived from v'_1 by deleting all nodes at odd depths;
 - If $v_1 = \langle r \rangle v'_1 \langle /r \rangle$ (with $v'_1 \in \text{NW}(\Sigma)$), then $v_2 = \langle r \rangle v'_2 \langle /r \rangle$, where v'_2 is derived from v'_1 by deleting all nodes at odd depths.
- All other strings are rejected by R .

The target language T contains all strings v of the following kinds:

- $v = \langle l \rangle v' \langle /l \rangle$, where the tree represented by v' has some path from root to leaf containing a label different from $\#$.
- $v = \langle r \rangle v' \langle /r \rangle$, where the tree represented by v' has some path from root to leaf

- containing a label $\#$, or
 - containing two subsequent labels from U^1 or two subsequent labels from U^2 , or
 - containing some label u^2 from U^2 followed either by u_{err} or by a u'^1 with $(u, u') \notin V$, or
 - containing some label from \hat{U} that is *not* part of a vertical error.
- $v = \langle x \rangle v' \langle /x \rangle$, where the tree represented by v' has some path from root to leaf
 - containing a label $\#$, or
 - containing two subsequent labels from U^1 or two subsequent labels from U^2 , or
 - containing some label u^2 from U^2 followed either by u_{err} or by a u'^1 with $(u, u') \notin V$, or
 - containing some label from \hat{U} that is *not* part of a vertical error
 - containing *no* vertical error, *no* label u_{err} and ending with a label different from u_f^1, u_f^2 .

It is relatively easy to see (but tedious to prove formally) that this construction is possible in polynomial time and that JULIET indeed has a winning strategy in G on w if and only if Player 1 has no winning strategy on the tiling instance \mathcal{I} . \square

For the upper bound, we first prove decidability in 2-EXPTIME for a “purely NWT-based” problem, to which we will later reduce JWIN.

Definition A.9. The *alternating iterated transduction* problem for non-deleting NWT is defined as the following decision problem:

AIT(NWT)	
Given:	A string $w \in \text{NW}(\Sigma)$, number k (given in unary), DNWA A , and pairs of non-deleting NWTs $(T_{1,0}, T_{1,1}), \dots, (T_{k,0}, T_{k,1})$
Question:	Is there an $i_1 \in \{0, 1\}$ such that for every $w_1 \in T_{1,i_1}(w)$ there exists $i_2 \in \{0, 1\}$ such that for every $w_2 \in T_{2,i_2}(w_1) \dots$ there exists $i_k \in \{0, 1\}$ such that for every $w_k \in T_{k,i_k}(w_{k-1})$ it holds that $w_k \in L(A)$?

Proposition A.10. $\text{AIT(NWT)} \in 2\text{-EXPTIME}$

Proof. We assume, without loss of generality, that each transducer $T_{j,i}$ (for $j \in [k], i \in \{0, 1\}$) is in normal form, i.e. that each (reading or ϵ -) transition of each transducer produces exactly one output tag (cf Lemma 5).

The idea behind this proof is to eliminate the existential quantification in the problem setting by constructing an NWT T that simulates in parallel *both* transducers $T_{j,1}$ and $T_{j,2}$ at each level $j \in [k]$. More concretely, T takes as input a string w and outputs a 2^k -tuple of strings, each component of which corresponds to a sequence of existential choices $i_1 \dots i_k \in \{0, 1\}^k$ of transducers, while the non-determinism in T simulates universal choice. It then holds (as we will prove after the construction of T) that the condition of AIT(NWT) is fulfilled if and only if for each possible transduct w' of w by T , at least one of

the 2^k component strings is contained in (an appropriate modification of) the target language $L(A)$.

To construct T , we construct from $T_{j,0}$ and $T_{j,1}$ for each level $j \in [k]$ a transducer T_j that takes as input a nested string w_{j-1} over an alphabet of 2^{j-1} -tuples of alphabet symbols and outputs a nested string w_j over 2^j -tuples, with the intuition being that T_j simulates one run of both $T_{j,0}$ and $T_{j,1}$ on each of the 2^{j-1} input strings encoded in w_{j-1} to produce a total of $2 \cdot 2^{j-1} = 2^j$ output strings, which are encoded in w_j . From all of the transducers T_j for all $j \in [k]$, we then use Proposition 7 to construct T as the transducer for $T_k \circ \dots \circ T_1$.

The main difficulty in the construction of each T_j is the fact that, while both $T_{j,0}$ and $T_{j,1}$ should be simulated for exactly one run on each of the components on the input string, all of these runs should be independent of each other. For instance, if T_2 wanted to simulate some run ρ of $T_{2,0}$ on the first and a different run ρ' of $T_{2,0}$ on the second input string component, ρ might start by reading the first tag from the input string while ρ' starts with an ϵ -transition and reads the first input tag afterwards. In such a situation, ρ' would produce an output before the output of ρ even starts. We therefore need to construct each T_j in such a manner that runs are *synchronised*, producing an output for *all* components of the output string, even if only *one* run of some transducer T_{j,i_j} calls for producing an output.

To address this problem, we introduce a special *blank* symbol $\sqcup \notin \Sigma$ and construct each T_j in such a way that reading transitions of T_j simulate synchronous reading transitions of $T_{j,0}$ and $T_{j,1}$ on *all* symbols of the next input tuple; ϵ -transitions of T_j , on the other hand, simulate only *one* of $T_{j,0}$ or $T_{j,1}$ on only *one* of the 2^{j-1} input components and output tuples consisting of exactly one symbol from Σ in the corresponding component and \sqcup symbols in all the others. To keep this construction consistent and allow for later transductions, this also means that all transducers $T_{j,i}$ ($j \in [k]$, $i \in \{0, 1\}$) have to be modified in such a way that, on reading an opening (closing) \sqcup tag, they always output an opening (closing) \sqcup tag and do not change states.

More formally, if (for each $j \in [k]$ and $i \in \{0, 1\}$) $T_{j,i} = (Q_{j,i}, P_{j,i}, P_{j,i}^\epsilon, \Sigma, \delta_{j,i}, q_{j,i}^0, F_{j,i})$, then let first $T'_{j,i}$ be an extension of $T_{j,i}$ to strings containing \sqcup tags that ignores and reproduces these tags as described above, i.e. $T'_{j,i} = (Q_{j,i}, P_{j,i} \cup \{p_\sqcup\}, P_{j,i}^\epsilon, \Sigma \cup \{\sqcup\}, \delta'_{j,i}, q_{j,i}^0, F_{j,i})$, where $p_\sqcup \notin P_{j,i}$ and $\delta'_{j,i}$ consists of $\delta_{j,i}$ extended by transitions $(q, \langle \sqcup \rangle, q, p_\sqcup, \langle \sqcup \rangle)$ and $(q, p_\sqcup, \langle \sqcup \rangle, q, \langle \sqcup \rangle)$ for each $q \in Q_{j,i}$. For any $w \in \text{NW}(\Sigma \cup \{\sqcup\})$, let $\text{Strip}(w) \in \text{NW}(\Sigma)$ denote the nested string obtained from w by deleting all \sqcup -labelled tags, and for any set $S \subseteq \text{NW}(\Sigma \cup \{\sqcup\})$, let $\text{Strip}(S) \stackrel{\text{def}}{=} \{\text{Strip}(w) \mid w \in S\}$. Then, it is clear that for any $w \in \text{NW}(\Sigma \cup \{\sqcup\})$, it holds that $\text{Strip}(T'_{j,i}(w)) = T_{j,i}(\text{Strip}(w))$.

Now we describe in detail the construction of T_j from $T'_{j,0}$ and $T'_{j,1}$. Let

$$T_j = (Q_j, P_j, P_j^\epsilon, \Sigma_j^{\text{in}} \cup \Sigma_j^{\text{out}}, \delta_j, q_j^0, F_j),$$

where the linear and hierarchical state sets as well as initial and final states simply derive from a 2^{j-1} -fold product construction of $T'_{j,0}$ and $T'_{j,1}$, i.e.

$$\begin{aligned} Q_j &= (Q_{j,0} \times Q_{j,1})^{2^{j-1}}, \\ P_j &= ((P_{j,0} \cup \{p_\sqcup\}) \times (P_{j,1} \cup \{p_\sqcup\}))^{2^{j-1}}, \\ q_j^0 &= (q_{j,0}^0 \times q_{j,1}^0)^{2^{j-1}}, \end{aligned}$$

and

$$F_j = (F_{j,0} \times F_{j,1})^{2^{j-1}}.$$

The working alphabet of T_j consists of input alphabet $\Sigma_j^{\text{in}} = (\Sigma \cup \{\sqcup\})^{2^{j-1}}$ and output alphabet $\Sigma_j^{\text{out}} = (\Sigma \cup \{\sqcup\})^{2^j}$. The construction of P_j^ϵ is similar, but has to be adjusted slightly; since ϵ -transitions of T_j are supposed to simulate $T_{j,0}$ or $T_{j,1}$ on only *one* component of the output string and the other components have to be filled in with \sqcup symbols, we add a new hierarchical ϵ -state $p_\sqcup^\epsilon \notin P_{j,0} \cup P_{j,1}$ and set $P_j^\epsilon = ((P_{j,0}^\epsilon \cup \{p_\sqcup^\epsilon\}) \times (P_{j,1}^\epsilon \cup \{p_\sqcup^\epsilon\}))^{2^{j-1}}$.

In accordance with the above intuition, we construct the transition relation δ_j of T_j . Reading transitions in δ_j (i.e. transitions that read one input symbol and produce one output symbol) are pretty much products of transitions from $\delta'_{j,0}$ and $\delta'_{j,1}$. That is, δ_j contains an opening transition which, for $\ell \in [2^j]$, starts with linear state q , ends in state q' and produces hierarchical state q in the ℓ -th position of the corresponding state tuples while reading $\langle a \rangle$ in the $\lfloor \frac{\ell}{2} \rfloor$ -th position of the input tuple and writing $\langle b \rangle$ in the ℓ -th position of the output tuple if the transition $(q, \langle a \rangle, q', p, \langle b \rangle)$ is in $\delta'_{j,0}$ (for odd ℓ) or in $\delta'_{j,1}$ (for even ℓ).⁵ Closing reading transitions in δ_j are constructed accordingly.

As for ϵ -transitions, δ_j contains an opening transition $(\bar{q}, \langle \epsilon \rangle, \bar{q}', \bar{p}, \bar{a})$ if there is an $\ell \in [2^j]$ such that (with \bar{x}_i denoting the i -th component of a tuple \bar{x})

- for all $\ell' \neq \ell$, it holds that $\bar{q}_{\ell'} = \bar{q}'_{\ell'}$, $\bar{p}_{\ell'} = p_\sqcup^\epsilon$ and $\bar{a}_{\ell'} = \sqcup$, and
- for $\bar{q}_\ell = q$, $\bar{q}'_\ell = q'$, $\bar{p}_\ell = p$ and $\bar{a}_\ell = a$, there is a transition $(q, \langle \epsilon \rangle, q', p, \langle a \rangle)$ in $\delta'_{j,0}$ (for odd ℓ) or in $\delta'_{j,1}$ (for even ℓ),

and accordingly for closing ϵ -transitions.

With this construction, T_j is indeed an NWT in normal form (i.e. fulfils the ϵ -consistency, well-formedness and synchronisation properties). This directly implies that T_j transduces well-nested strings over Σ_j^{in} into well-nested strings over Σ_j^{out} . We now identify nested strings over tuples of alphabet symbols with tuples of nested strings as follows: For some j , let $\bar{w} = \bar{w}^1 \dots \bar{w}^n$ be a well-nested string over $(\Sigma \cup \{\sqcup\})^{2^j}$, and for each $\bar{w}^i \in ((\Sigma \cup \{\sqcup\})^{2^j})$, let \bar{w}_ℓ^i denote its ℓ -th component interpreted as an opening tag (and analogously for closing tags). Then the nested string \bar{w}_ℓ is defined as $\bar{w}_\ell = \bar{w}_\ell^1 \dots \bar{w}_\ell^n$. It is clear that, if $\bar{w} \in \text{NW}((\Sigma \cup \{\sqcup\})^{2^j})$, then $\bar{w}_\ell \in \text{NW}(\Sigma \cup \{\sqcup\})$ for each ℓ .

Identifying strings of tuples with tuples of strings in this way, it is easy (if tedious) to prove that, for each j and each string $\bar{w} \in \text{NW}(\Sigma_j^{\text{in}})$, interpreted as a 2^{j-1} -tuple $(\bar{w}_1, \dots, \bar{w}_{2^{j-1}})$ of nested strings, it holds that $\text{Strip}(T_j(w)) = \prod_{\ell=1}^{2^{j-1}} T_{j,0}(\text{Strip}(\bar{w}_\ell)) \times T_{j,1}(\text{Strip}(\bar{w}_\ell))$, where the Strip operator is applied component-wise, i.e. $\text{Strip}((\bar{w}_1, \dots, \bar{w}_n)) = (\text{Strip}(\bar{w}_1), \dots, \text{Strip}(\bar{w}_n))$. In other words, disregarding \sqcup tags, every run of T_j on a 2^{j-1} -tuple of nested words simulates one run each of $T_{j,0}$ and $T_{j,1}$ on each of its component strings, and all combinations of such component runs can be simulated by a run of T_j .

By a simple induction argument, it follows that each transducer of the form $T_j \circ \dots \circ T_1$ completely describes all possible series of existential choices up to

⁵At first glance, it may seem more intuitive to associate even positions with $T_{j,0}$ and odd positions with $T_{j,1}$, but seeing as ℓ is a number between 1 and 2^j which will later be encoded as $\ell - 1$ by a binary sequence of length j , the association described here is indeed the more useful one.

the j -th level. More precisely, denoting by $S_\ell = \{\bar{w}_\ell \mid \bar{w} \in S\}$ the set of all ℓ -th component strings in a set S of tuples of nested strings, we get that for every $j \leq k$, $w \in \text{NW}(\Sigma)$ and $\ell \in [2^j]$, it holds that $\text{Strip}((T_j \circ \dots \circ T_1(w))_\ell) = T_{j,i_j} \circ \dots \circ T_{1,i_1}(w)$, where $i_1 \dots i_j \in \{0,1\}^j$ is the binary representation of the number $\ell - 1$.

To use the transducer $T = T_k \circ \dots \circ T_1$ for solving the alternating iterated transduction problem as described initially, we now need to prove that the defining property of $\text{AIT}(\text{NWT})$ (i.e. “There is an $i_1 \in \{0,1\}$ such that for every $w_1 \in T_{1,i_1}(w)$...”) is equivalent to the following: For every $\bar{w} \in T(w)$, there is an $\ell \in [2^k]$ such that $\text{Strip}(\bar{w}_\ell) \in L(A)$. We denote this property by (*). As we have already seen, the existence of ℓ in property (*) is equivalent to the existence of an index sequence $i_1 \dots i_k \in \{0,1\}^k$ representing $\ell - 1$ in binary, so we basically need to prove that the alternation between existential and universal choices in the defining property of $\text{AIT}(\text{NWT})$ is equivalent to a single universal choice of iterated transducts for each possible index sequence of existential choices, followed by a single existential choice of index sequence.

The proof of this equivalence is by induction; the central step is proving, for any set S of nested strings, the equivalence of

$$\forall \bar{w} \in T_{j-1} \circ \dots \circ T_1(w) \exists \ell \in [2^{j-1}] \exists i_j \in \{0,1\} \forall w' \in T_{j,i_j}(\text{Strip}(\bar{w}_\ell)) : w' \in S$$

and

$$\forall \bar{w}' \in T_j \circ T_{j-1} \circ \dots \circ T_1(w) \exists \ell' \in [2^j] : \text{Strip}(\bar{w}'_{\ell'}) \in S,$$

which follows primarily from the fact that T_j simulates both $T_{j,0}$ and $T_{j,1}$ as was shown before.

To sum up the proof thus far, we have constructed from an $\text{AIT}(\text{NWT})$ instance a transducer T such that the original instance is a positive one if and only if for the input string w it holds that all transducts in $T(w)$ have a component string that is in $L(A)$ when stripped of all \sqcup -labelled tags. It remains to be seen how we can check for this property in doubly exponential time.

To that end, let A' be a modification of A that ignores \sqcup -labelled tags, constructed from A using a similar construction to that one for each T_{j,i_j} . From A' , we can construct a DNWA B that gets as input a nested string of tags over 2^k -tuples of symbols from $\Sigma \cup \{\sqcup\}$, simulates a copy of A' on each of the 2^k components and accepts if and only if at least one of its components accepts. It then holds that the original $\text{AIT}(\text{NWT})$ instance is a positive one if and only if $T(w) \subseteq L(B)$, which yields an instance of a type checking problem. Each level transducer T_j is of size at most $\mathcal{O}(|T_{j,0}| \cdot |T_{j,1}|^{2^k})$, therefore T is of doubly exponential size, and so is B . Since the type checking problem with target DNWA is decidable in PTIME by Theorem 13(b), this yields a 2-EXPTIME algorithm for $\text{AIT}(\text{NWT})$, as was to be proven. \square

A reduction to $\text{AIT}(\epsilon\text{-NWT})$, along with Proposition A.8 now proves Theorem 15.

Theorem 15 (restated). *For the class of replay-free games with NWT, JWIN is 2-EXPTIME-complete.*

Proof. The lower bound was proven as Proposition A.8. We prove a matching upper bound by reduction to $\text{AIT}(\epsilon\text{-NWT})$, which is in 2-EXPTIME according to Proposition A.10.

Let $G = (\Sigma, \Gamma, R, T)$ be a game with NWT replacement, and let w be an input string for G . Let furthermore $k \leq \frac{|w|}{2}$ be the number of occurrences of closing tags from $\langle \Gamma \rangle$ in w .

The idea behind the reduction to $\text{AIT}(\epsilon\text{-NWT})$ is taking k rounds of alternating transduction, where the j -th round (with transducers $T_{j,0}, T_{j,1}$) corresponds to the replay-free subgame on the j -th function symbol in w (in left-to-right order). The choice between transducers $T_{j,0}$ and $T_{j,1}$ models JULIET's choice between Read and Call; to that end, $T_{j,0}$ basically does not change its input string at all, while $T_{j,1}$ simulates the replacement transducer R on the substring that JULIET chose to be replaced. The only minor technical difficulty in this construction is the fact that, in the game G , the transducer R only rewrites the called substring, while each $T_{j,i}$ rewrites the entirety of the current string. This difficulty can be solved by some minor modifications, which we will now examine.

The input string w' for $\text{AIT}(\epsilon\text{-NWT})$ is derived from w by replacing, in left-to-right order, each substring $\langle f \rangle v \langle /f \rangle$ of w by $\langle j \rangle \langle f \rangle v \langle /f \rangle \langle /j \rangle$, where $\langle /f \rangle$ is the j -th closing function tag in w and $j \notin \Sigma$ for each $j \in [k]$. In other words, the substring on which JULIET has to make her j -th strategy decision is encapsulated in j -tags.

For each $j \in [k]$, the transducer $T_{j,0}$ simply deletes the $\langle j \rangle$ and $\langle /j \rangle$ tags from its input, leaving it otherwise unchanged. The transducer $T_{j,1}$, on the other hand, also directly outputs its input until it reaches the $\langle j \rangle$ tag. It deletes this tag and then starts simulating the replacement transducer R . Once $T_{j,1}$ reaches the $\langle /j \rangle$ tag, it deletes that tag as well and stops its simulation, rejecting its input if R has not reached an accepting state. Afterwards, $T_{j,1}$ simply outputs its input again. Note that the simulation of R in $T_{j,1}$ will never receive as input any tags with labels not in Σ , as all such tags have a label strictly less than j and have therefore already been removed by earlier transductions.

The target DNWA A for $\text{AIT}(\epsilon\text{-NWT})$ is simply the target DNWA $A(T)$ of G .

It is easy to see that k , w' and each $T_{j,0}$ for $j \in [k]$ can be computed from w in polynomial time, as can each $T_{j,1}$ from R . As the alternating transduction simulates the replay-free game, it is also clear that JULIET has a replay-free winning strategy on w in G if and only if the constructed instance for $\text{AIT}(\epsilon\text{-NWT})$ is a positive one, which concludes the reduction. \square

Theorem 17 (restated). *For the class of replay-free games with depth-bounded NWT, JWIN is EXPSPACE -complete.*

We prove the lower and upper bounds of this theorem as separate propositions.

Proposition A.11. *For the class of replay-free games with depth-bounded NWT, JWIN is EXPSPACE -hard.*

Proof. (sketch) This lower bound proof uses the problem $\text{EXPONENTIAL CORRIDOR TILING}$: Given a tiling instance consisting of a tile set U , vertical and horizontal constraints V , H , initial and final tile u_i, u_f and unary number n , is there a tiling of width $2^n - 1$ and arbitrary height?

From any input tiling instance, we construct a game $G = (\Sigma, \Gamma, R, T)$ with depth-bounded NWT R and an input word w such that JULIET has a winning strategy on w in G if and only if there exists *no* valid tiling of width $2^n - 1$, i.e.

we reduce from the complement of EXPONENTIAL CORRIDOR TILING. Since EXPONENTIAL CORRIDOR TILING is EXPSPACE-complete, so is its complement.

The basic idea behind the construction of G is similar to that used in the proof of Proposition A.8. The input string is $w = \langle r \rangle \langle o_n \rangle \langle e_n \rangle \cdots \langle o_1 \rangle \langle e_1 \rangle \langle s \rangle \langle /s \rangle \langle /e_1 \rangle \langle /o_1 \rangle \cdots \langle /e_n \rangle \langle /o_n \rangle \langle /r \rangle$. JULIET is first supposed to Call $\langle /s \rangle$, allowing ROMEO to respond with the standard nested string encoding \hat{v}_0 of a flat string $v_0 \in (U \cup \{\#\})^*$ which is supposed to encode a valid tiling t of width $2^n - 1$ in the standard way (i.e. lines of t are concatenated and separated by $\#$ symbols). We can fix the transducer R in such a way that v_0 is always a concatenation of horizontally correct substrings (not necessarily of length $2^n - 1$) separated by $\#$ symbols. After \hat{v}_0 is given, JULIET then plays Call on either e_1 or o_1 , prompting ROMEO to delete either all even-numbered or all odd-numbered positions in v_0 and yielding the standard nested string encoding \hat{v}_1 of the resulting flat string v_1 . Continuing this process further (i.e. sequentially calling exactly one of e_i or o_i for each $i \in [n]$) eventually yields a nested string \hat{v}_n which encodes what is supposed to be a single column in the tiling given by ROMEO. This string is then checked for vertical correctness by the target DNWA. In this way, JULIET has a winning strategy on w in G if and only if every tiling candidate that can be provided by ROMEO contains some (horizontal or encoding) error, i.e. if there is no valid tiling of the desired width. \square

Similar to the proof of Theorem 15, the upper bound in Theorem 17 is proven by a reduction to (a variant of) the alternating iterated transduction problem: Given a string $w \in \text{NW}(\Sigma)$, number k (given in unary), DNWA A , and pairs of depth-bounded NWTs $(T_{1,0}, T_{1,1}), \dots, (T_{k,0}, T_{k,1})$, is there an $i_1 \in \{0, 1\}$ such that for every $w_1 \in T_{1,i_1}(w)$ there exists $i_2 \in \{0, 1\}$ such that for every $w_2 \in T_{2,i_2}(w_1) \dots$ there exists $i_k \in \{0, 1\}$ such that for every $w_k \in T_{k,i_k}(w_{k-1})$ it holds that $w_k \in L(A)$?

Proposition A.12. The alternating iterated transduction problem for depth-bounded NWT is in EXPSPACE.

Proof. Basically, the exponential-space algorithm solving this problem follows the same idea as the 2-EXPTIME algorithm from the proof of Proposition A.10; here, the bounded depth of input transducers ensures that the construction and type checking of transducers T_j (for each j) and T from that proof may be simulated on-the-fly by a co-nondeterministic exponential-space algorithm \mathcal{A} .

More specifically, it can be easily proven by induction that, in any run ρ of an NWT receiving as input a string of depth d_{in} and outputting a string of depth at most d_{out} , the sequence of hierarchical states in any configuration occurring in ρ has length at most $d_{\text{in}} + d_{\text{out}}$.

Since all input transducers are depth-bounded, we can assume without loss of generality that they have a common upper bound d on their output depth. Then, for each $j > 1$, each $T_{j,i}$ has an input *and* output depth bounded by d , since $T_{j,i}$ receives as its input the output of some $T_{j-1,i'}$. On the other hand, $T_{1,0}$ and $T_{1,1}$ also have an output depth bounded by d and an input depth bounded by $|w|$, which is also fixed for fixed input strings w . Similarly, the DNWA A receives as inputs only outputs of $T_{k,0}$ or $T_{k,1}$, so we may restrict our attention to configurations of A with a hierarchical state sequence of length at most d .

The idea behind the algorithm \mathcal{A} , then, is to traverse the input string w from left to right and, on each input tag, co-nondeterministically guess transducts for each combination of follow-up transducers and check that at least one resulting final transduct is in $L(A)$, thus simulating the transducer T from the proof of Proposition A.10 and checking that each transduct from $T(w)$ is accepted by the NWA B from that proof.

The algorithm \mathcal{A} is organised into k layers, with the j -th layer (for $j \in [k]$) simulating the workings of transducer T_j from the proof of Proposition A.10, which takes as input 2^{j-1} -tuples of symbols from $\Sigma \cup \{\sqcup\}$ and outputs 2^k -tuples of such symbols by guessing transitions of both $T_{j,0}$ and $T_{j,1}$ on each component of the input tuple. The algorithm \mathcal{A} does this simulation based on a single tuple of symbols for each layer (called that layer's *active input tuple*) – once layer $j-1$ produces an output tuple, this tuple becomes the active input for layer j , and \mathcal{A} continues its simulation with layer j , producing an output to become the active input tuple for layer $j+1$ by either consuming the active input tuple of layer j or by an ϵ -transition (which leaves the active input tuple for layer j unchanged). Once the active input tuple of some layer j has been consumed, control passes back to layer $j-1$. The bottom layer k , instead of producing further active input tuples, directly simulates the effect of its output on 2^k modified copies of the DNWA A (i.e. simulates the NWA B from the proof of Proposition A.10).

Some extra care has to be taken regarding deletions. The proof sketch so far (as well as the construction for Proposition A.10) assumes that simulated transitions are synchronised in such a way that, when an input tuple is consumed, each component of that tuple produces two components of the output tuple. This assumption obviously does not hold in the presence of deleting transitions. For this reason, we allow active input tuples to contain *null positions* labelled \perp , which do not induce any transition on the corresponding transducers once the tuple is consumed.

In the sequel, as in the proof of Proposition A.10, assume that all input transducers are in normal form as per Lemma 5, and for each $i \in \{0,1\}$ and $j \in [k]$, let $T'_{j,i}$ be an extension of $T_{j,i}$ to strings containing \sqcup tags that ignores and reproduces these tags, and let A' be an analogous extension of A . We now describe the algorithm \mathcal{A} in more detail.

For each j , the following information is stored for layer j :

- A 2^j -tuple of configurations of $T'_{j,0}$ (in odd positions) and $T'_{j,1}$ (in even positions), each consisting of a linear state and a sequence of at most $d+|w|$ (for $j=1$) or $2d$ (for $j>1$) hierarchical states. These configurations are initialised with the starting configuration of $T'_{j,0}$ or $T'_{j,1}$.
- A 2^{j-1} -tuple of symbols from $(\Sigma \cup \{\sqcup\}) \cup \langle \Sigma \cup \{\sqcup\} \rangle \cup \{\perp\}$, where either all non- \perp components are opening tags or all are closing tags (called the *active input tuple* of layer j). The non- \perp components of each active input tuple gives the next input symbols to be consumed by (2^{j-1} copies of) $T_{j,0}$ and $T_{j,1}$. The active input tuple of layer j is initialised to $\perp^{2^{j-1}}$.

Additionally, \mathcal{A} stores the following global information:

- A counter $\ell \in \{0, \dots, |w|\}$ denoting the last position of the input string w that has been read, initialised to $\ell = 0$.

- A 2^k -tuple of configurations of A' , each consisting of a single linear state and a sequence of up to d hierarchical states. These configurations are initialised with the starting configuration of A' .
- A layer counter $j \in \{0, \dots, k\}$ denoting the current *active layer* being processed (with layer 0 corresponding to the input string itself). This counter is initialised to $j = 0$.

All of this information can obviously be stored in exponential space.
The algorithm \mathcal{A} now proceeds as follows:

1. If $j = 0 \dots$
 - (a) ...and $\ell < |w|$, then the input has not yet been completely read. In this case, \mathcal{A} sets the active input (2^0 -)tuple of layer 1 to the $(\ell + 1)$ -th symbol of w , $\ell := \ell + 1$, and $j := 1$.
 - (b) ...and $\ell = |w|$, then the input has been processed completely, and \mathcal{A} halts.
2. If $0 < j < k \dots$
 - (a) ...and the active input tuple of layer j is *not* equal to $\perp^{2^{j-1}}$, then layer j still has some input to be processed. In this case, \mathcal{A} guesses co-nondeterministically which of the following two steps to perform:
 - i. \mathcal{A} processes the input of layer j , i.e. for each non- \perp position $i \in [2^{j-1}]$ of the active input tuple, \mathcal{A} guesses a transition of $T'_{j,0}$ with the input symbol from that position starting at the configuration in position $2i - 1$ of the configuration tuple for layer j , and updates that configuration accordingly; similarly, \mathcal{A} guesses a transition for $T'_{j,1}$ with position $2i$ of the configuration tuple. The outputs of these transitions are written to positions $2i - 1$ and $2i$ of a new 2^j -tuple t . For any \perp -position $i \in [2^{j-1}]$, positions $2i - 1$ and $2i$ of t are then filled with \perp markers. Afterwards, the active input tuple of layer $j + 1$ is set to t , the active input tuple of layer j is reset to $\perp^{2^{j-1}}$, and $j := j + 1$.
 - ii. \mathcal{A} performs an ϵ -transition. To that end, \mathcal{A} guesses an index $i \in [2^j]$. If i is odd, \mathcal{A} guesses an opening (resp. closing) ϵ -transition for $T'_{j,0}$ available in the configuration at position $(i + 1)/2$ of the configuration tuple and updates that configuration accordingly, while configurations at all other positions remain unchanged. It then writes the output of that transition to position $(i + 1)/2$ of a new 2^j -tuple t and fills all other positions of t with $\langle \sqcup \rangle$ (resp. $\langle / \sqcup \rangle$) symbols. If i is even, \mathcal{A} proceeds analogously with $T'_{j,1}$ instead of $T'_{j,0}$ and $i/2$ instead of $(i + 1)/2$. Finally, \mathcal{A} sets the active input tuple of layer $j + 1$ to t and $j := j + 1$ *without* resetting the active input tuple of layer j .
 - (b) ...and the active input tuple of layer j is $\perp^{2^{j-1}}$, i.e. layer j currently has no input waiting to be processed. In this case, \mathcal{A} guesses co-nondeterministically which of the following two steps to perform:
 - i. \mathcal{A} performs an ϵ -transition, as described under 2.a.ii.

- ii. \mathcal{A} ends its processing of layer j and sets $j := j - 1$.
- 3. If $j = k$, then \mathcal{A} basically proceeds as described under 2., with the only difference being that any time \mathcal{A} has created an output tuple t of size 2^k , instead of setting $j := j + 1$, \mathcal{A} directly aggregates t onto the configuration 2^k -tuple for A' , simulating, for each $i \in [2^k]$, a transition of A' starting from the i -th component of the configuration tuple, consuming the i -th component of t and storing the resulting configuration in the i -th position of the configuration tuple for A' .

Finally, once \mathcal{A} halts, it accepts if at least one of the following conditions is met:

- One of the 2^k stored configurations for A' is accepting. In this case, there is some sequence of existential choices of transducers such that, for the universal choices made co-nondeterministically by \mathcal{A} , the final transduct is in $L(A)$.
- For some $j \in [k]$, the configuration tuple for layer j contains some non-accepting configuration. In this case, the co-nondeterministic choices taken by \mathcal{A} have lead to an incorrect transduction being performed, i.e. the corresponding run of \mathcal{A} should not be counted against the acceptance condition.

To show correctness of \mathcal{A} , it needs to be proven that there exists a non-accepting run of \mathcal{A} if and only if for all $i_1 \in \{0, 1\}$ there exists $w_1 \in T_{1, i_1}(w)$ such that for all $i_2 \in \{0, 1\}$... for all $i_k \in \{0, 1\}$ there exists $w_k \in T_{k, i_k}(w_{k-1})$ with $w_k \notin L(A)$.

The proof for this is rather technical, but its basic idea is as follows: For the “only if” direction, we extract from an accepting run of \mathcal{A} each “witness string” w_j inductively based on the sequence i_1, \dots, i_j of universal choices and prior witness strings w_1, \dots, w_{j-1} by taking the output string produced in component $i + 1$ of layer j , where $i \in \{0, \dots, 2^j - 1\}$ is the number represented by the binary encoding $i_1 \dots i_j$. Using the construction of \mathcal{A} , it is then easy to see that $w_j \in T_{j, i_j}(w_{j-1})$, and that $w_k \notin L(A)$. Similarly, for the “if” direction we can construct a run of \mathcal{A} from witness strings given universal choices of indices in $\{0, 1\}$, which is non-accepting because all transductions are performed correctly and, regardless of universal choices, the resulting final string w_k is not in $L(A)$.

Finally, as stated above, \mathcal{A} requires only co-nondeterministic exponential space and can therefore be simulated by an EXPSPACE algorithm. \square

Proof of Theorem 17. The lower bound was proven as Proposition A.11. For the upper bound, we reduce JWIN to the alternating iterated transduction problem for depth-bounded transducers using the same reduction as in the proof of Theorem 15.

Some care needs to be taken with this reduction, as the reduction from Theorem 15 constructs, from an input string w and a game G whose replacement transducer has output depth bounded by some constant d , an instance for the alternating iterated transduction problem whose transducers have a depth bound $d + |w|$; that is, the depth bound for the alternating iterated transduction problem actually depends on the size of the input. However, the proof of

Proposition A.12 shows that the algorithm given there still only takes exponential space, even if the depth bound of transducers depends on the input, since in this case, the algorithm keeps track of (exponentially many) configurations whose size is still at most polynomial in the input size. In this way, Proposition A.12 still yields an EXPSPACE upper bound for JWIN. \square

Proofs for Section 5

This section gives proofs for the results of Section 5. Since Lemma 6 applies in this setting, we generally prove lower bounds for deleting ϵ -free NWT and upper bounds for non-deleting ϵ -free NWT.

Theorem 18 (restated). *For the class of games with ϵ -free NWT and unbounded replay, JWIN is undecidable.*

Proof. The proof is by reduction from the halting problem for Turing machines with an empty input. From a given Turing machine M with working alphabet Σ and state set Q , we construct a game G and string w such that JULIET has a winning strategy on w in G if and only if M halts on an empty input. Without loss of generality, we assume that M always moves its head to the left-most used tape cell before halting.

The basic idea behind the reduction is encoding configurations of M by strings nested below a root $r \notin \Sigma$ and using Call moves by JULIET to $\langle r \rangle$ to simulate moves of M . The input string is $w = \langle r \rangle \hat{v}_0 \langle r \rangle$, where \hat{v}_0 represents the initial configuration of M , and any time JULIET plays Call on $\langle r \rangle$ in some string $\langle r \rangle \hat{v}_i \langle r \rangle$ (with \hat{v}_i representing some configuration of M), that string gets replaced by $\langle r \rangle \hat{v}_{i+1} \langle r \rangle$, where \hat{v}_{i+1} represents the successor configuration of the one represented by \hat{v}_i . The target language of G is constructed to contain all strings of the form $\langle r \rangle \hat{v}_h \langle r \rangle$, where \hat{v}_h represents a halting configuration of M . This way, JULIET has a winning strategy of Call depth k on the input string if and only if M halts on the empty input within at most k steps.

We represent configurations of M in the same way as in the proof of Theorem 14: a flat string $(x_1, -) \cdots (x_{k-1}, -)(x_k, q)(x_{k+1}, -) \cdots (x_m, -)$ over the alphabet $\Sigma \times (Q \cup \{-\})$ denotes that the content of M 's working tape is $x_1 \cdots x_m$, with the head of M being on the tape's k -th cell and M being in state q , and these flat strings are represented as nested strings using the standard nested string representation. Again, by our assumption on the shape of M 's halting configurations, the flat string encoding of a halting configuration is of the form $(\Sigma \times \{h\})(\Sigma \times \{-\})^*$ for the halting state h of M .

From M , we can easily construct a ϵ -free NWT implementing M 's transition function on configurations represented in this way.⁶ A DNWA accepting all strings $\langle r \rangle v \langle r \rangle$ in which v represents a halting configuration for M is similarly easy to construct. Finally, it is clear that JULIET has a winning strategy on the input string w using at most k Call moves if and only if M reaches a halting configuration from its initial configuration within at most k steps, which completes the proof. \square

Throughout the rest of this section, let $\text{Exp}(k, n)$ be the k -fold exponential tower function in n , defined recursively by $\text{Exp}(0, n) = n$ and $\text{Exp}(k, n) = 2^{\text{Exp}(k-1, n)}$ for all integers $k > 0$ and $n \geq 0$.

Lemma A.13. An input string of length $2 \cdot (n + 2k - 1)$ can be transformed into a string of length $2 \cdot \text{Exp}(k, n)$ by a game of Call depth 2 with deterministic ϵ -free NWT replacement.

⁶Note that this ϵ -free NWT is functional, so G is basically a “solitaire” game for JULIET, where ROMEO does not get to make any choices. In fact, the replacement relation can even be implemented using a deterministic finite-state transducer that inserts at most two symbols with each transduction.

Proof. Choose as input a string of the form $\langle k-1 \rangle \langle k-2 \rangle \cdots \langle 1 \rangle \langle c_0 \rangle^n \langle c_1 \rangle \cdots \langle c_k \rangle \langle /c_k \rangle \cdots \langle /k-1 \rangle$. The tree represented by this string is obviously a path of length $n+2k-2$. Play proceeds in k rounds as follows: In round $i \in [k]$, JULIET plays Call on each node labelled c_{i-1} in bottom-up (i.e. left-to-right) order. Each such Call move deletes the called c_{i-1} node and doubles the number of c_i nodes below it (i.e. replaces each $\langle c_i \rangle$ by $\langle c_i \rangle \langle c_i \rangle$ and $\langle /c_i \rangle$ by $\langle /c_i \rangle \langle /c_i \rangle$). Afterwards, if $i < k$, JULIET plays Call on the node labelled i , which deletes that node, attaches its child path to its $i+1$ -labelled parent and allows JULIET to play again on its child path, since this is the first time JULIET has played Call on the node labelled i . By an induction argument, it is easy to show that at the conclusion of round i , the current string contains exactly $2 \cdot \text{Exp}(i, n)$ tags labelled c_i (half of them opening, the other half closing tags). \square

Lemma A.13 is an integral part in proving the following lower bound.

Proposition A.14. For each $k \geq 1$, it holds that for the class of games with ϵ -free NWT, and Call depth bounded by 2, JWIN is hard for CO- k -NEXPTIME.

Proof. Let $k \geq 1$. We show CO- k -NEXPTIME-hardness by reduction from the complement of the k -NEXPTIME-complete problem k -EXPTILING of, given a tile set U , vertical and horizontal constraints V, H , initial and final tile u_i, u_f and unary number n , determining whether there exists a valid tiling of height $\text{Exp}(k, n)$ and width $\text{Exp}(k, n) - 1$.

We construct from an instance of k -EXPTILING a game G and a string w such that JULIET has a winning strategy in G starting at w if and only if there exists *no* valid tiling of the given size. We first give a rough overview over the game before describing the construction of G and w in detail.

Play proceeds in three phases:

- In the first phase, the construction from Lemma A.13 is used to transform the polynomial-sized input string into a *seed string* of size $\mathcal{O}(\text{Exp}(k-1, n))$. This phase is deterministic in the sense that we will design the replacement transducer and target language in a way such that neither JULIET nor ROMEO get to make any choices in phase 1.
- In phase 2, ROMEO constructs from this seed string a *tiling candidate*, i.e. a (representation of a) potential tiling of size $\text{Exp}(k, n) \times (\text{Exp}(k, n) - 1)$. In this phase, JULIET still doesn't get to make any choices; all she is supposed to do is to Call certain nodes in order to allow ROMEO to construct the tiling candidate. By fixing the replacement transducer in the proper way, we can ensure that this tiling candidate has no horizontal errors and starts with the initial tile.
- In the final phase, JULIET tries to show that the tiling candidate ROMEO constructed in phase 2 does not represent a valid tiling. To this end, she repeatedly forces ROMEO to delete either all even or all odd columns in his tiling candidate until only a single column is left; this column is then checked for vertical errors by the target language automaton.

Note that, during all of the three phases, we require JULIET to stick to a certain “game plan”, in which she only calls certain nodes in a pre-determined order. This can be enforced by constructing the replacement transducer and

target language automaton in an appropriate way. We will first examine phase-by-phase how the game G is constructed, assuming JULIET's compliance, and later describe how G has to be modified to prevent deviations from the game plan.

Let $v_{k,n}$ be the sequence of opening tags from the proof of Lemma A.13, i.e. $v_{k,n} = \langle k-1 \rangle \langle k-2 \rangle \cdots \langle 1 \rangle \langle c_0 \rangle^n \langle c_1 \rangle \cdots \langle c_k \rangle$, and let $\overline{v_{n,k}}$ be the complementary sequence of closing tags. The input string w , then, is of the form

$$w = \langle \text{start2} \rangle \langle \text{mv} \rangle \langle \text{dbl} \rangle \langle \text{cp1} \rangle v_{k-1,n} \langle r \rangle \langle p \rangle \langle /p \rangle \langle /r \rangle \overline{v_{k-1,n}} \langle /cp1 \rangle \langle /dbl \rangle \langle /mv \rangle \langle /start2 \rangle.$$

The purpose of each tag will be explained when it is first used in the game.

At the start of phase 1, JULIET uses the procedure described in the proof of Lemma A.13 to rewrite w into the following string of $(k-1)$ -fold exponential size:

$$\langle \text{start2} \rangle \langle \text{mv} \rangle \langle \text{dbl} \rangle \langle \text{cp1} \rangle \langle c_{k-1} \rangle^\ell \langle r \rangle \langle p \rangle \langle /p \rangle \langle /r \rangle \langle c_{k-1} \rangle^\ell \langle /cp1 \rangle \langle /dbl \rangle \langle /mv \rangle \langle /start2 \rangle,$$

where $\ell = \text{Exp}(k-1, n)$. Afterwards, JULIET plays her first Call on the node labelled cp1 (where “cp” stands for “copy”), allowing her a replay on $\langle c_{k-1} \rangle^\ell \langle r \rangle \langle /r \rangle \langle c_{k-1} \rangle^\ell$. In this replay, she calls every c_{k-1} in left-to-right order; each such Call replaces the called c_{k-1} by c'_{k-1} , replaces $\langle r \rangle$ by $\langle r \rangle \langle d \rangle \langle e \rangle$ and replaces $\langle /r \rangle$ by $\langle /e \rangle \langle /d \rangle \langle /r \rangle$. After this rewriting, the r -labelled node has a child path consisting of ℓ alternating d - and e -nodes.

Next, JULIET plays Call on the node labelled dbl (for “double”), which rewrites the c'_{k-1} -labelled path into a c'_{k-1} -labelled path of double length, i.e. the rewriting transducer replaces each $\langle c'_{k-1} \rangle$ by $\langle c'_{k-1} \rangle \langle c'_{k-1} \rangle$ and each $\langle /c'_{k-1} \rangle$ by $\langle /c'_{k-1} \rangle \langle /c'_{k-1} \rangle$. With her next call to mv (for “move”), JULIET gets another replay on this path of length 2ℓ , calling each c'_{k-1} node in left-to-right order, which causes ROMEO to delete that node and insert a single c (for “create”) node below the bottom p -labelled node. Finally, JULIET plays Call on the root node labelled start2 to start the second phase of the game; as his reply to this Call, ROMEO deletes the start2-node and inserts a node labelled u_i as a leaf at the very bottom of the path. This ends phase 1 and leaves JULIET to play a replay-free game on the string

$$w_1 = \langle r \rangle \langle \langle d \rangle \langle e \rangle \rangle^\ell \langle p \rangle \langle c \rangle^{2\ell} \langle u_i \rangle \langle /u_i \rangle \langle c \rangle^{2\ell} \langle /p \rangle \langle \langle /e \rangle \langle /d \rangle \rangle^\ell \langle /r \rangle.$$

In the second phase, ROMEO is supposed to set up a tiling candidate. This candidate will be encoded within the leaves of the current string, i.e. in siblings of the initial u_i -labelled leaf. To this end, JULIET calls each of the c -nodes in bottom-up order. On each such call, ROMEO deletes the called node and doubles the number of leaves in the current string. More precisely, the replacement transducer may replace each $\langle /u \rangle$ (for some $u \in U$) either by $\langle /u \rangle \langle u' \rangle \langle /u' \rangle$ for some $u' \in U$ with $(u, u') \in H$, or by $\langle /u \rangle \langle \# \rangle \langle / \# \rangle$ for a divider symbol $\# \notin U$. Since each Call to a c -node doubles the length of the current tiling candidate, it is clear that when all c -nodes have been called (and phase 2 ends), the current string is of the form

$$w_2 = \langle r \rangle \langle \langle d \rangle \langle e \rangle \rangle^\ell \langle p \rangle v_t \langle /p \rangle \langle \langle /e \rangle \langle /d \rangle \rangle^\ell \langle /r \rangle,$$

where v_t , the final tiling candidate, is a string of nesting depth zero consisting a total of $2^{2\ell} = (2^\ell)^2 = \text{Exp}(k, n)^2$ pairs of corresponding opening and closing tags with labels from $U \cup \{\#\}$, beginning with $\langle u_i \rangle \langle /u_i \rangle$.

We interpret the tiling candidate v_t as the concatenation of $\text{Exp}(k, n)$ lines of length $\text{Exp}(k, n)$ each. To encode a valid tiling of size $\text{Exp}(k, n) \times (\text{Exp}(k, n) - 1)$, we expect v_t to be of the form $((\langle U \rangle \langle /U \rangle)^{\text{Exp}(k, n) - 1} \#)^{\text{Exp}(k, n)}$, i.e. we expect ROMEO to use the symbol $\#$ as a line separator only.

It should be clear from the construction that the string v_t represents a concatenation of several (possibly empty) horizontally correct substrings of tiles, separated by $\#$, the first of which starts with u_i . The tiling candidate encoded by v_t may, however, still contain one or more of the following types of errors:

- *Vertical error*: Two vertically adjacent tiles u, u' with $(u, u') \notin V$;
- *Incorrect line lengths*: Strictly more or less than $\text{Exp}(k, n) - 1$ symbols from U between two subsequent $\#$;
- *Incorrect final tile*: The last symbol from U in v_t is not u_f .

The main observation needed for this part of the reduction is the following: if a tiling candidate does not represent a correct tiling, then at least one of these errors can be found by examining just a single column of the tiling candidate. If a tiling candidate contains a vertical error, then there is a column containing two subsequent tiles u, u' with $(u, u') \notin V$; if some line length is incorrect, then the last column contains some symbol other than $\#$, or there is a $\#$ in a column that is *not* the last column; and finally, if the tiling candidate does not end with u_f , then the next-to-last column (i.e. column number $\text{Exp}(k, n) - 1$) does not end with u_f . It is easy to see that, once a single column of v_t has been isolated, all three of these conditions can easily be checked using a polynomial-sized DNWA.

JULIET's task on the string w_2 therefore consists of isolating a single column containing an error. If and only if she manages to do so, she wins the game (i.e. the target DNWA checking for the existence of one of the above kinds of errors accepts).

To isolate a single column, JULIET plays Call moves on all nodes labelled d (for “destroy”) in bottom-up order. Each such Call removes the called d -node and its child, and forces the replacement transducer to either delete all even-numbered columns or all odd-numbered columns by deleting every other node in v_t . JULIET makes the choice of whether to delete all even-numbered or all odd-numbered columns by playing either Read or Call on the e -labelled node just below the d -node she is to call next; a Read move leaves the label e (“even”) intact, causing the replacement transducer to delete all even-numbered columns on JULIET's Call to the d -node above, while a Call move relabels e into o (“odd”), causing all odd-numbered columns to be deleted analogously. Each such deletion step halves the number of remaining columns, which means that after ℓ deletion moves, only a single column of length $\text{Exp}(k, n)$ remains of v_t .

From the above considerations concerning error types, it is clear that some additional information needs to be tracked through the deletion phase, to determine whether the column chosen by JULIET is the last, next-to-last, or some other column. This is the purpose of the node labelled p (“position”), which is rewritten depending on the sequence of JULIET's choices of even and odd columns:

- To reach the last column, JULIET must successively remove only odd-numbered columns. Therefore, on the first Call to a d -node with o -child

(and p -grandchild), p is rewritten into l (“last”). Any Call to a d -node with e -child and l -grandchild rewrites l into s (“standard”, i.e. the column to be checked is not a special case), while calling a d -node with o -child leaves the label l as is.

- To reach the next-to-last column, JULIET must remove all even-numbered columns in the first step and successively remove only odd-numbered columns after that. Therefore, on the first Call to a d -node with e -child (and p -grandchild), p is rewritten into x (“neXt-to-last”). Any Call to a d -node with o -child and x -grandchild leaves the label x intact, while a Call to a d -node with e -child and x -grandchild relabels x to s .

Using the label of the rewritten p -node as an indicator, a polynomial-sized DNWA can now easily check whether a column chosen by JULIET contains some error. Assuming that JULIET is restricted to strategies that follow the described order of Call moves, it is straightforward (if tedious) to prove that JULIET has a winning strategy if and only if there does not exist a valid tiling of size $\text{Exp}(k, n) \times (\text{Exp}(k, n) - 1)$. If such a tiling exists, ROMEO can give its encoding as v_t and deny JULIET the opportunity to find any errors, no matter which column she isolates, and if no such tiling exists, then any tiling candidate given by ROMEO necessarily contains at least one error, which JULIET can then point out.

Finally, we examine how JULIET may be restricted to play only according to the game plan described above. The basic idea behind this is to construct the replacement transducer and target DNWA in such a way that any deviation from the game plan causes JULIET to immediately and irrevocably lose the game, thus ensuring that any winning strategy, should one exist, sticks to the game plan.

The construction of the target language so far already ensures that JULIET does not leave any undesired uncalled nodes behind, i.e. since target strings may only have tags from $U \cup \{\#, r, l, x, s\}$, JULIET loses automatically if, for instance, there are any uncalled d -nodes left behind in the final string.

The only thing that still needs to be ensured is that JULIET does not skip any calls. For instance, we could imagine JULIET trying to cheat in phase 1 by playing Call on some i -labelled node without having called all nodes labelled c_{i-1} below it, or in phase 3 by calling some d -labelled node while there are still uncalled c -labelled nodes. There are numerous further situations like these, but they are all handled in the same way: on any Call to a function symbol, the nested substrings supposed to be rewritten due to that Call are required to be of a specific form; for instance, on a Call to some i -labelled node in phase 1, the substring nested below it has to be of the form

$$\langle c_i \rangle^* \langle c_{i+1} \rangle \cdots \langle c_k \rangle \langle r \rangle \langle p \rangle \langle /p \rangle \langle /r \rangle \langle /c_k \rangle \cdots \langle /c_{i+1} \rangle \langle /c_i \rangle^*.$$

If the substring below the called function node is not of the desired form (for instance due to remaining c_{i-1} -nodes in the previous example), the replacement transducer constructed according to the ideas laid out above will not have an accepting run on this incorrect substring, therefore reject that substring, and, according to the semantics of NWT games, ROMEO will win the game immediately. This shows that we can safely assume JULIET to be restricted to the game plan laid out above.

Finally, it is relatively easy (but again rather tedious) to prove that the replacement transducer described above can be constructed to be of polynomial size in $|U|$, n and k . \square

Theorem 19 (restated). *For the class of games with ϵ -free NWT and Call depth bounded by $d \geq 2$, JWIN is decidable, but not decidable in elementary time.*

Proof. Decidability follows from the fact that, due to the restriction to ϵ -free NWT, there are only finitely many possible replacements for each substring that JULIET plays Call on. This, combined with the finite Call depth, means that all strategies for JULIET and all possible plays for each strategy can be enumerated in finite time.

The non-elementary lower bound on complexity is implied by Proposition A.14: Assume that, for some $d \geq 2$, there exists a k such that $\text{JWIN}(\mathcal{G}_d)$ is decidable in k -fold exponential time, where \mathcal{G}_d denotes the class of games with ϵ -free NWT and Call depth d . It then follows (by a trivial reduction) that $\text{JWIN}(\mathcal{G}_2)$ is also in k -EXPTIME, and therefore in $\text{CO-}k$ -NEXPTIME. However, by Proposition A.14, $\text{JWIN}(\mathcal{G}_2)$ is hard for $\text{CO-}(k+1)$ -NEXPTIME, which yields a contradiction to the nondeterministic time hierarchy theorem [7]. \square

Theorem 20 (restated). *For the class of replay-free games with ϵ -free NWT, JWIN is CO-NEXPTIME -complete.*

Proof. The lower bound follows as in the proof for Proposition A.14, omitting the first phase of the game constructed there and setting $k = 1$ in the second and third phase.

The co-nondeterministic exponential-time algorithm yielding a matching upper bound is conceptually very straightforward: It moves through the input string from left to right, recursively trying out all possible strategy decisions for JULIET while guessing universally ROMEO's strategy decisions.

To formalise this algorithm, we use the shorthand notation $[u, v]$ for game positions, where $uv \in \text{NW}(\Sigma)$, with u denoting the substring that has already been processed and v the substring that is yet to be played on (including the closing tag on which JULIET is to move next). For a string $u\langle/a\rangle \in \hat{\Sigma}^*$, let $\text{LAST}(u\langle/a\rangle)$ denote the (unique) rooted substring of $u\langle/a\rangle$ ending at $\langle/a\rangle$. The following algorithm $\text{CHECKWIN}(G, [u, v])$ then tests whether JULIET has a replay-free winning strategy in game $G = (\Sigma, \Gamma, R, T)$ starting at position $[u, v]$.

As the algorithm $\text{CHECKWIN}(G, [u, v])$ directly mimics the gameplay according to G from position $[u, v]$, a simple induction argument suffices to prove that $\text{CHECKWIN}(G, [u, v])$ accepts if and only if JULIET has a replay-free winning strategy in G from $[u, v]$; therefore, $\text{CHECKWIN}(G, [\epsilon, w])$ accepts if and only if $w \in \text{JWin}^1(G)$. It remains to be shown that CHECKWIN indeed runs in co-nondeterministic exponential time.

To that end, we first examine the maximum size of positions used as inputs for recursive calls of CHECKWIN (where the size of a position $[u, v]$ is defined as $|uv|$). On some input $[u, v]$, CHECKWIN can only increase the size of $[u, v]$ through the transduction in lines 14 and 15. Since R is a NWT without ϵ -transitions, any of its outputs on some input x may only have size $c \cdot |x|$ for some

Algorithm 1 CHECKWIN($G, [u, v]$)

```
1: if  $v = \epsilon$  then
2:   if  $u \in T$  then
3:     Accept
4:   else
5:     Reject
6: if  $v = tv'$  for  $t \in \langle \Sigma \rangle \cup (\langle \Sigma \rangle \setminus \langle \Gamma \rangle)$  then
7:   // JULIET may not make a strategy choice on  $t$ ; move to the right.
8:   Return CHECKWIN( $G, [ut, v]$ )
9: if  $v = \langle f \rangle v'$  for  $f \in \Gamma$  then
10:  // Try out both strategy options for JULIET; Read first.
11:  if CHECKWIN( $G, [u\langle f \rangle, v]$ ) accepts then
12:    Accept
13:  else
14:    Guess universally a transduct  $u_t \in R(\text{LAST}(u\langle f \rangle))$ 
15:     $u' \leftarrow u\langle f \rangle$  with  $\text{LAST}(u\langle f \rangle)$  replaced by  $u_t$ 
16:    if CHECKWIN( $G, [u', v']$ ) accepts then
17:      Accept
18:    else
19:      Reject
```

constant c depending only on R . This in turn means that the size of positions may only increase by a factor of c for each recursive call to CHECKWIN.

Obviously, as each recursive call to CHECKWIN removes one symbol from the right side of a position and insertions only occur on the left side of positions, the recursion depth of CHECKWIN on an input position $[\epsilon, w]$ is at most w . This in turn means that input positions for recursive calls to CHECKWIN may be of size at most $|w| \cdot c^{|w|}$.

Finally, since each recursive call branches into at most two further calls of CHECKWIN, the algorithm's recursion tree has at most exponentially many nodes in the size of w . This implies that the membership test in line 2 and the co-nondeterministic choice in line 14 are executed at most exponentially many times on at most exponentially long strings. Since membership testing both for NWA and for NWT is in PTIME, this yields an exponential upper bound on the running time of CHECKWIN. \square

Theorem 21 (restated). *For the class of games with ϵ -free NWT, Call depth bounded by $d \geq 1$ and Call width bounded by $k \geq 1$, JWIN is co-NEXPTIME-complete.*

Proof. As replay-free games always have bounded Call width, the lower bound carries over directly from Theorem 20.

The algorithm for the upper bound is almost the same as the one for Theorem 20, simply going through the input string from left to right, trying out all possible strategies for JULIET and guessing co-nondeterministically replacement strings for ROMEO. The only difference is that, in this case, it also needs to track how much of the allowed Call width and depth in the current substring has been used up already.

As in the proof of Theorem 20, the correctness of this algorithm is obvious as it basically just simulates gameplay.

To prove that the running time and amount of non-determinism required by the algorithm is at most exponential, we again examine the maximum size of occurring strings and of the decision tree for JULIET for any fixed counter-strategy of ROMEO, i.e. the recursion tree in any run of the algorithm.

For string sizes, we again note that, for any replacement NWT R and string w , all strings in $R(w)$ are of size at most $s \cdot |w|$ for some constant s only depending on R . Furthermore, any replacement substring v resulting from a Call move of depth $r < d$ allows for at most k further Call moves of depth $r + 1$ (each of which may be further played on with Call width k if $r + 1 < d$). Each of these depth- $(r + 1)$ substrings may be of size at most $s^k \cdot |v|$ (in case all k allowed calls go into re-transducing v in some fashion). In total, this means that increasing Call depth by 1 increases the size of occurring strings by a multiplicative factor exponential in k . Therefore, with Call depth d and Call width k , the final string after a play on some input string w is at most of size $|w| \cdot 2^{\mathcal{O}(dk)}$, and this size is also an upper bound on the size of each single transduct (and therefore also on the amount of non-determinism required any time a transduct is guessed).

Since each Call move of depth r enables at most k further calls of depth $r + 1$, and there are at most $|w|$ possible Call positions of depth 0 in any input string w , JULIET may play at most $c_{\max} \stackrel{\text{def}}{=} |w| \cdot k^d$ Call moves in any play on w .

To reach an upper bound on the number of positions JULIET has available to place calls on during an entire play, we use the above upper bound of $|w| \cdot 2^{\mathcal{O}(dk)}$ on the size of working strings and assume (as a generous estimate) that the *entire* string is replaced by a different string of length $|w| \cdot 2^{\mathcal{O}(dk)}$ every time JULIET plays Call on some position. Then, the total number ℓ_{\max} of positions JULIET can place calls on is bounded by the total number of positions that ever become available during the entire game, which is at most the maximum string length times the maximum number of available Call moves, i.e. $\ell_{\max} = (|w| \cdot 2^{\mathcal{O}(dk)}) \cdot (|w| \cdot k^d) = |w| \cdot 2^{\mathcal{O}(dk)}$.

We can now give an upper bound on the size of the decision tree for JULIET for a given counter-strategy by ROMEO. Assume nodes of this tree to be labelled with positions that JULIET has to make a strategy decision on, and outgoing edges to be labelled either Read or Call. Then, each root-to-leaf path has length equal to at most ℓ_{\max} and contains at most c_{\max} edges labelled Call, with all other edges being labelled Read. Since the sequence of Read- and Call-edges taken from the root forms a unique address in this tree, the number of different paths in the tree is at most the number of strings of length ℓ_{\max} over $\{\text{Read}, \text{Call}\}$ containing at most c_{\max} occurrences of Call, which is in $\ell_{\max}^{\mathcal{O}(c_{\max})}$. Together with ℓ_{\max} being the maximum length of paths, this implies that the decision tree has at most $\ell_{\max}^{\mathcal{O}(c_{\max})} = |w| \cdot 2^{\mathcal{O}(|w| \cdot dk^{d+1})}$ nodes.

Since d and k are fixed, this means that the size of the recursion tree for the co-nondeterministic algorithm simulating play is at most exponential, and since each node of this recursion tree requires at most exponential computation time and non-determinism, the algorithm indeed runs in co-nondeterministic exponential time, as was to be proven. \square

Theorem 22 (restated). *For the class of games with ϵ -free NWT, Call depth bounded by d and Call width including input bounded by k , JWIN is*

- (a) CO-NP-complete for $d \geq 1$ and $k \geq 2$,
- (b) CO-NP-complete for $d \geq 2$ and $k \geq 1$, and
- (c) in PTIME for $d = k = 1$.

Proof. The CO-NP upper bound uses almost exactly the same algorithm as Theorem 21, going through the input string from left to right, trying out all possible strategies for JULIET, guessing co-nondeterministically replacement strings for ROMEO and tracking Call depth and width along the way. The only difference to that algorithm lies in the fact that Call width is also tracked for the input string, which changes the analysis from the proof of Theorem 21 as follows.

With fixed bounds d on Call depth and k on Call width, the maximum number of Call moves that JULIET can play is $c_{\max} = k^d$, which is constant. With each Call move increasing the size of the current string by at most a multiplicative constant s (depending only on the replacement transducer), this means that the maximum size of strings is $\ell_{\max} = |w| \cdot s^{c_{\max}}$ for any input string w , which is linear in $|w|$. In analogy to the arguments from the proof of Theorem 21, this means that JULIET may perform calls on at most c_{\max} out of $c_{\max} \cdot \ell_{\max} = \mathcal{O}(|w|)$ positions, implying that the decision tree has at most $\ell_{\max}^{\mathcal{O}(c_{\max})}$ paths of length ℓ_{\max} each, and therefore size $\mathcal{O}(|w|^c)$ for some constant c . Since all strings that need to be processed or nondeterministically guessed at each node of the recursion tree are of polynomial length, this implies a co-nondeterministic polynomial time complexity.

The lower bounds in (a) and (b) follow by reductions from the problem CO-3-SAT: Given a propositional formula φ in conjunctive normal form with three literals per clause, is φ unsatisfiable? This problem is the complement of the well-known NP-complete problem 3-SAT, and therefore complete for CO-NP.

The reductions work as follows. Let $\varphi = C_1 \wedge \dots \wedge C_m$ be a 3-CNF formula over variables x_1, \dots, x_n with clauses C_1, \dots, C_m . We construct from φ a game $G = (\Sigma, \Gamma, R, T)$ and input string w such that ROMEO has a winning strategy on w in G if and only if φ is satisfiable; the basic idea behind the reduction is that JULIET's first Call to the input string allows ROMEO to pick a variable assignment α for φ , while JULIET's second Call is supposed to mark a clause that is not satisfied by α (if one exists). In this way, JULIET has a winning strategy if and only if every possible assignment that ROMEO can choose does not satisfy all of the clauses in φ . Using this idea, JULIET needs to perform exactly two Call moves; where and how these calls should be performed depends on whether the game is replay-free with Call width 2 (for (a)) or has Call depth 2 and Call width 1 (for (b)).

For both reductions, the input string w is of the form

$$w = \langle r \rangle \langle C_1 \rangle \dots \langle C_m \rangle \langle V \rangle \langle x_1 \rangle \langle /x_1 \rangle \dots \langle x_n \rangle \langle /x_n \rangle \langle /V \rangle \langle /C_m \rangle \dots \langle /C_1 \rangle \langle /r \rangle,$$

i.e. its tree representation is a path with labels r, C_1, \dots, C_m, V (in top-down order), with the V -labelled node having as children leaves labelled x_1, \dots, x_n .

For the reduction in (a), the set of function symbols are $\Gamma = \{V, C_1, \dots, C_m\}$. JULIET's first Call is supposed to be on $\langle /V \rangle$, which relabels V to V' and each x_i to either 0_i or 1_i (for $i \in [n]$), at ROMEO's choice; afterwards, she calls exactly one $\langle /C_j \rangle$ (with $j \in [m]$), which relabels the called node to \hat{C}_j and leaves the

rest of the string unchanged. If JULIET plays Call on some $\langle /C_j \rangle$ without having called $\langle /V \rangle$ first, the replacement transducer rejects its input string and JULIET loses the game immediately. The target language accepts all strings containing exactly one node labelled with a symbol $\hat{C}_j \in \{\hat{C}_1, \dots, \hat{C}_m\}$ such that the clause C_j is not satisfied by the variable assignment encoded below V .

For (b), the set of function symbols is $\Gamma = \{r, C'_1, \dots, C'_m\}$. JULIET's first Call (the only one she can perform in the input string) is to $\langle /r \rangle$, which relabels r to r' , each C_j to C'_j (for $j \in [m]$) and each x_i to either 0_i or 1_i (for $i \in [n]$), at ROMEO's choice; afterwards, JULIET is supposed to call exactly one $\langle /C'_j \rangle$, relabelling it to \hat{C}_j . The target language accepts all strings rooted with r' that contain exactly one node labelled with a symbol $\hat{C}_j \in \{\hat{C}_1, \dots, \hat{C}_m\}$ such that the clause C_j is not satisfied by the variable assignment encoded below V .

It is easy to see that both reductions can be computed in polynomial time and that in both cases, JULIET has a winning strategy in G if and only if φ is not satisfiable.

The PTIME upper bound in (c) is quite simple. Let $G = (\Sigma, \Gamma, R, T)$ be a game and w an input string; we now need to find out whether exactly one of the function symbols occurring in w can be called to yield a string in the target language. For any fixed occurrence of a function symbol in w , it is easy to construct a NWT R' that simulates R on the substring rooted at the chosen function symbol and otherwise leaves w unchanged. A type checking test against the target DNWA can then determine in polynomial time (cf. Theorem 13 (b)) whether calling that function symbol leads to a string in the target language. Since there are less than $|w|$ occurrences of function symbols in w , polynomially many such type checking tests suffice to determine whether JULIET has a winning strategy in G on w that plays at most one Call move. \square

Proofs for Section 6

Theorem 23 (restated). *For the class of games with relabelling transducers and unbounded replay, JWIN is EXPTIME-complete.*

Proof. The upper bound uses a trivial alternating polynomial-space algorithm that moves through the input string in a left-to-right order (resetting its focus as necessary after Call moves), guesses existentially for each closing tag a move for JULIET and, in case of a Call, guesses universally a relabelling chosen by ROMEO and applies it to the input string. As relabellings are generally of linear size, they can easily be guessed on polynomial space, and the verification whether a guessed relabelling is indeed consistent with the replacement transducer is feasible in polynomial time by Theorem 11.

The lower bound follows by a reduction from the membership problem for linear bounded alternating Turing machines, which is complete for $\text{APSPACE} = \text{EXPTIME}$ [5].

Let M be a linear bounded ATM with state set Q and working alphabet Σ (i.e. on input w , M uses at most $|w|$ tape cells in any computation). Assume without loss of generality that any non-halting state of M is either existential or universal, that the transition relation for M has exactly two transitions for each state and tape symbol (i.e. each non-halting configuration of M has exactly two successor configurations), that the initial state of M is universal and that M always moves its head to the left-most tape position before halting.

We construct from M and a given input string w a game G and input string w' such that JULIET has a winning strategy on w' in G if and only if M accepts w . To this end, we use a similar technique as for Theorem 18: We represent configurations of M by strings rooted at a function symbol $r \notin \Sigma$ and have each Call by JULIET to $\langle /r \rangle$ initiate a transition of M , updating M 's configuration by the replacement transduction.

The main difference to the proof of Theorem 18 is that we have to take alternation into account. We simulate the alternation in M by strategy choices of JULIET and ROMEO, with JULIET choosing existential and ROMEO choosing universal transitions. Universal choice can simply be encoded into the replacement transducer, such that when JULIET initiates a transition of M , ROMEO chooses which of the two possible successor configurations to rewrite the current string to. A little more care has to be taken with existential choice, as JULIET may not select any rewritings but can only choose whether or not a substring should be rewritten.

To allow JULIET to choose between transitions, we extend the nested word representation of M 's current configuration by a special *flag substring*, which may be either $\langle 0 \rangle \langle /0 \rangle$ or $\langle 1 \rangle \langle /1 \rangle$ or $\langle 2 \rangle \langle /2 \rangle$. This flag should be 0 if the current configuration is universal or halting, and initially set to 1 once an existential configuration is reached. If it is 1, JULIET has the option to have ROMEO rewrite it to 2 by way of a Call move, or leave it as is with a Read move. Once JULIET then initiates a transition of M , the flag indicates which of the two possible successor configuration is reached by the rewriting, i.e. ROMEO does not get any choice but rewrites the current string based on the flag's value.

Once again, we represent configurations of M as in the proof of Theorem 14: a flat string $(x_1, -) \cdots (x_{k-i}, -)(x_k, q)(x_{k+1}, -) \cdots (x_m, -)$ over the alphabet $\Sigma \times (Q \cup \{-\})$ denotes that the content of M 's working tape is $x_1 \cdots x_m$, with

the head of M being on the tape's k -th cell and M being in state q , and these flat strings are represented as nested strings using the standard nested string encoding. By our assumption on the shape of M 's halting configurations, the flat string encoding of an accepting configuration is of the form $(\Sigma \times \{q_+\})(\Sigma \times \{-\})^*$ for the accepting state q_+ of M .

The input string w' for the game G is constructed as $w' = \langle r \rangle \langle 0 \rangle \langle /0 \rangle \hat{v}_0 \langle /r \rangle$, where \hat{v}_0 is the nested string representation of M 's initial configuration with input w (which is universal by our assumption above, hence the 0 flag). The game G is over the alphabet $(\Sigma \times (Q \cup \{-\})) \cup \{r, 0, 1, 2\}$ with function symbols $\Gamma = \{r, 1\}$.

The replacement transducer simulates a transition of M at a Call on $\langle /r \rangle$ as described above. Such a transducer can be computed from M 's transition relation in polynomial time. It is also easy to construct in polynomial time a target DNWA accepting all strings of the form $\langle r \rangle \langle 0 \rangle \langle /0 \rangle v \langle /r \rangle$ where v represents an accepting configuration of M . It follows from the above considerations that JULIET has a winning strategy on w' in G if and only if M accepts w . \square

Theorem 24 (restated). *For any $k \geq 1$, for the class of games with relabelling transducers and bounded Call depth k , JWIN is PSPACE-complete.*

Proof. The upper bound again uses the trivial alternating algorithm that moves through the input string in a left-to-right order, guesses existentially for each closing tag a move for JULIET and, in case of a Call, guesses universally a relabelling chosen by ROMEO and applies it to the input string. As in the proof of Theorem 23, relabellings can be guessed and verified in polynomial time; additionally, since the game has bounded Call depth k , strategy decisions and relabellings are guessed at most $k \cdot |w|$ times, which yields a polynomial time bound for the alternating algorithm.

The lower bound for $k = 1$ follows directly from the PSPACE lower bound proof for games with fixed replacement languages without transducers ([14], Proposition 10(b)), as that proof only requires relabelling. \square

Theorem 25 (restated). *For the class of games with functional relabelling transducers and unbounded replay, JWIN is PSPACE-complete.*

Proof. As for Theorems 23 and 24, the upper bound uses the trivial algorithm simulating the game with functional relabelling transducers by moving through the input string in a left-to-right order (resetting its focus as necessary after Call moves), guessing existentially for each closing tag a move for JULIET and, in case of a Call, also guessing the relabelling chosen by ROMEO and applying it to the input string. Since there is only at most a single possible rewriting for each string input into the relabelling transducer and relabellings are of linear size in the input, this is a nondeterministic polynomial-space algorithm witnessing membership of JWIN in NPSPACE = PSPACE.

The lower bound for games with deterministic relabelling transducers is proven similarly to the one in Theorem 23 by reduction from the membership problem for linear bounded *deterministic* Turing machines. The simulation of a TM by a game works as in the proof of Theorem 18, as that proof also just requires a deterministic transducer. For linear bounded TMs, a relabelling transducer suffices, since no additional tape cells beyond those provided by the input are ever inserted.

Since all deterministic NWTs are also functional, the upper bound also applies to deterministic relabelling NWTs and the lower bound also to functional relabelling NWTs, which proves the claim. \square

Theorem 26 (restated). *For any $k \geq 1$, for the class of games with functional relabelling transducers and bounded Call depth k , JWIN is NP-complete.*

Proof. The upper bound for functional relabelling transducers uses the same nondeterministic algorithm as the one used for the upper bound of Theorem 25; the only difference is that, since the input game has Call depth k , the algorithm has to guess nondeterministically and verify at most $k \cdot |w|$ strategy choices and relabellings, each of which may be done nondeterministically in polynomial time. This yields the desired NP upper bound.

We prove the lower bound for replay-free games with deterministic relabelling transducers by a reduction from the 3SAT problem: Given a propositional formula $\varphi = C_1 \wedge \dots \wedge C_m$ over variables x_1, \dots, x_n where each clause C_j is a disjunction of exactly three literals, is there an assignment $\alpha : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ such that φ evaluates to 1 under α ?

We construct from φ a game $G = (\Sigma, \Gamma, R, T)$ and string $w \in \text{NW}(\Sigma)$ such that JULIET has a winning strategy on w in G if and only if φ is satisfiable. The basic idea behind the reduction is that the string w contains *variable substrings*, on which JULIET chooses an assignment α by using Read and Call moves, and a *clause substring*, which models the structure of φ and which gets rewritten after JULIET's choice of variable assignment into a form that allows a target DNWA to check whether α satisfies φ .

More concretely, the clause substring is

$$w_\varphi = \langle C_1 \rangle \langle \ell_{1,1} \rangle \langle / \ell_{1,1} \rangle \langle \ell_{1,2} \rangle \langle / \ell_{1,2} \rangle \langle \ell_{1,3} \rangle \langle / \ell_{1,3} \rangle \langle / C_1 \rangle \cdots \langle C_m \rangle \langle \ell_{m,1} \rangle \langle / \ell_{m,1} \rangle \langle \ell_{m,2} \rangle \langle / \ell_{m,2} \rangle \langle \ell_{m,3} \rangle \langle / \ell_{m,3} \rangle \langle / C_m \rangle,$$

where for each $i \in [m]$, $j \in [3]$ and $k \in [n]$,

$$\ell_{i,j} = \begin{cases} x_k, & \text{if } x_k \text{ is the } j\text{-th literal of } C_i \\ \bar{x}_k, & \text{if } \neg x_k \text{ is the } j\text{-th literal of } C_i. \end{cases}$$

The input string w is constructed inductively; we set $w = w_1$, where for each $i \in [n+1]$,

$$w_i = \begin{cases} \langle r \rangle \langle y_i^0 \rangle \langle / y_i^0 \rangle w_{i+1} \langle / r \rangle, & \text{for } i \in [n] \\ w_\varphi, & \text{for } i = n+1. \end{cases}$$

Play on w is supposed to proceed as follows: JULIET first goes through the variable substrings of the form $\langle y_i^0 \rangle \langle / y_i^0 \rangle$, choosing whether or not to call each $\langle / y_i^0 \rangle$ in turn. On a Call, y_i^0 gets relabelled to y_i^1 . In this way, JULIET chooses a variable assignment α , with a remaining y_i^0 interpreted as $\alpha(x_i) = 0$ and a rewritten y_i^1 interpreted as $\alpha(x_i) = 1$.

Afterwards, JULIET is supposed to call each $\langle / r \rangle$ in turn. On each such Call, the relabelling transducer T takes the valuation y_i^0 (or y_i^1 respectively) of the variable immediately following the corresponding $\langle r \rangle$ and rewrites all occurrences of x_i in w_φ into 0 and all \bar{x}_i into 1 (or x_i into 1 and \bar{x}_i into 0, respectively). After all literals in w_φ have been rewritten into 0 or 1, the target DNWA simply needs to check whether, for each $j \in [m]$, the substring enclosed in C_j tags contains at least one substring $\langle 1 \rangle \langle / 1 \rangle$.

More formally, the game G uses the alphabet $\Sigma = \{x_i, \overline{x_i}, y_i^0, y_i^1 \mid i \in [n]\} \cup \{C_j \mid j \in [m]\} \cup \{r\}$ with function symbols $\Gamma = \{r\} \cup \{y_i^0 \mid i \in [n]\}$.

The relabelling transducer R behaves as follows:

- R rewrites each input of the form $\langle y_i^0 \rangle \langle /y_i^0 \rangle$ into $\langle y_i^1 \rangle \langle /y_i^1 \rangle$.
- On an input of the form $\langle r \rangle \langle \ell \rangle \langle / \ell \rangle v \langle /r \rangle$ (for $\ell \in \{y_i^0, y_i^1 \mid i \in [n]\}$), R memorises ℓ in its state and performs a relabelling on v :
 - If $\ell = y_i^0$ for some $i \in [n]$, then R relabels each x_i in v to 0 and each $\overline{x_i}$ to 1, and
 - If $\ell = y_i^1$ for some $i \in [n]$, then R relabels each x_i in v to 1 and each $\overline{x_i}$ to 0.

The target language T contains all strings w' of the form of w_1 as defined above, with the following modifications:

- w' does not contain any labels from $\{x_i, \overline{x_i} \mid i \in [n]\}$, and
- each $\langle C_j \rangle$ tag (for $j \in [m]$) is followed by at least one $\langle 1 \rangle \langle /1 \rangle$ substring before the corresponding $\langle /C_j \rangle$ tag.

The construction of G and w from φ is obviously possible in polynomial time. Furthermore, by the above consideration on the construction and verification of assignments for φ , it is easy to see that JULIET has a winning strategy on w in G if and only if φ is satisfiable. \square

Theorem 27 (restated). *For the class of write-once games with functional relabelling transducers, JWIN is in PTIME.*

Proof. The crucial insight for this proof is the fact that games with functional replacement transducers are essentially solitaire games for JULIET— the result of any Call on some substring is uniquely determined by that substring, with no choice for ROMEO.

We utilise this fact by constructing from a given game $G = (\Sigma, \Gamma, R, T)$ a (generally non-functional) relabelling NWT R_J such that for each input string $w \in \text{NW}(\Sigma)$, the image $R_J(w)$ is exactly the set of all strings that w can be rewritten into by some left-to-right sequence of Read and Call moves by JULIET. Checking for the existence of a winning strategy for JULIET on w then simply amounts to checking whether $R_J(w) \cap T \neq \emptyset$.

The NWT R_J is constructed from R by a simple modification: In its standard mode of operation, R_J simply reproduces its input. Before any opening tag, however, R_J may choose nondeterministically to start simulating R beginning with the next opening tag, rewrite the input substring up until the corresponding closing tag and then return to simply reproducing its input (or starting another simulation of R).

Now, for any $w, w' \in \text{NW}(\Sigma)$ it holds that $w' \in T(w)$ if and only if there is a write-once strategy for JULIET rewriting w into w' , which can be proven by a somewhat involved induction on the structure of w and w' . The two main insights required for this proof are the facts that (i) due to JULIET having complete information and R being functional, JULIET can make her decision whether to Read or Call already on *opening* tags instead of the corresponding closing tags, and (ii) since R_J is a relabelling transducer, the relabelling of a

closing tag according to R_J is already determined at the corresponding opening tag, which enforces the corresponding rewriting for JULIET to be write-once.

The construction of R_J is obviously possible in polynomial time. Furthermore, given R_J and an input string w , by Corollary 8 and Lemma 9 we can construct in polynomial time a polynomial-size ϵ -NWA deciding $R_J(w)$. Intersection nonemptiness with the target language T can then be checked for in polynomial time due to Lemma A.5 and Theorem A.6(a). \square